

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1



BÁO CÁO BÀI TẬP IOT VÀ ỨNG DỤNG

ĐỀ TÀI: XÂY DỰNG HỆ THỐNG CẢM BIẾN IOT

Nhóm học phần: 16

Sinh viên: Bùi Ngọc Vũ

Mã sinh viên: B22DCCN910

Giảng viên hướng dẫn: Nguyễn Quốc Uy

Hà Nội, 2025

PHỤ LỤC

LỜI CẢM ƠN.....	5
CHƯƠNG I: GIỚI THIỆU CHUNG	6
1. Bối cảnh và lý do chọn đề tài.....	6
2. Làm gì?	7
3. Phát triển như thế nào?	7
4. Phạm vi nghiên cứu và triển khai.....	8
5. Ý nghĩa và đóng góp.....	9
CHƯƠNG II: THIẾT KẾ HỆ THỐNG.....	10
1. Phân tích yêu cầu hệ thống.....	10
1.1. Phân tích yêu cầu hệ thống	10
1.2. Yêu cầu phi chức năng	10
2. Thiết kế phần cứng.....	10
2.1. ESP32 – DEVKIT	10
2.2. Module cảm biến nhiệt độ, độ ẩm DHT11.....	14
2.3. Module cảm biến ánh sáng BH1750.....	18
2.4. Module Relay.....	21
2.5. Sơ đồ khối phần cứng	24
2.6. Sơ đồ đấu dây	26
3. Thiết kế giao diện	28

3.1. Giao diện trang chủ	29
3.2. Giao diện dữ liệu cảm biến	30
3.3. Giao diện lịch sử hoạt động.....	31
3.4. Giao diện hồ sơ.....	32
4. Thiết kế cơ sở dữ liệu	34
4.1. Lược đồ ERD.....	34
4.2. Mối quan hệ giữa các bảng	34
5. Sequence Diagram.....	35
5.1. Sequence 1: HomeFrm hiển thị thiết bị, dữ liệu cảm biến, điều khiển ON/OFF.....	35
5.2. Sequence 2: Hiển thị dữ liệu cảm biến với tìm kiếm và lọc	39
CHƯƠNG III. XÂY DỰNG HỆ THỐNG	43
1. Arduino.....	43
1.1. Thiết lập và triển khai thư viện	43
1.2. Cấu hình hệ thống.....	43
2. Cấu trúc hệ thống.....	49
3. Test API.....	50
3.1. Lấy danh sách thiết bị.....	50
3.2. Gửi lệnh điều khiển.....	51
3.3. Tìm kiếm có điều kiện.....	52

4. Giao diện	53
4.1. Giao diện trang chủ	53
4.2. Giao diện dữ liệu cảm biến.....	53
4.3. Giao diện lịch sử hoạt động.....	54
4.4. Giao diện hồ sơ	54

LỜI CẢM ƠN

Trước tiên, em xin được bày tỏ sự trân trọng và biết ơn sâu sắc đối với thầy Nguyễn Quốc Uy, giảng viên dẫn dắt em trong suốt thời gian qua. Do chưa có nhiều kinh nghiệm làm đề tài nhiều cũng như những hạn chế về kiến thức, trong BTL của em chắc chắn sẽ không tránh khỏi những thiếu sót. Rất mong nhận được sự nhận xét, ý kiến đóng góp và phê bình từ thầy để dự án của em được hoàn thiện hơn.

Lời cuối cùng, em xin kính chúc thầy nhiều sức khỏe, thành công và hạnh phúc.

Em xin chân thành cảm ơn!

CHƯƠNG I: GIỚI THIỆU CHUNG

1. Bối cảnh và lý do chọn đề tài

Trong những năm gần đây, cùng với sự phát triển mạnh mẽ của cuộc cách mạng công nghiệp 4.0, **Internet of Things (IoT)** đã và đang trở thành một trong những xu hướng công nghệ nổi bật. IoT cho phép kết nối hàng tỷ thiết bị, cảm biến, máy móc và hệ thống thông tin với nhau thông qua mạng Internet, tạo thành một hệ sinh thái thông minh. Nhờ đó, các hoạt động trong đời sống và sản xuất có thể được giám sát, điều khiển và tối ưu hóa một cách hiệu quả hơn.

Tại Việt Nam, ứng dụng IoT đang ngày càng được chú trọng và triển khai trong nhiều lĩnh vực khác nhau, từ nông nghiệp thông minh (giám sát độ ẩm, nhiệt độ đất, tự động tưới tiêu) cho đến quản lý giao thông, tòa nhà, nhà máy sản xuất hay hệ thống y tế. Trong đó, các hệ thống cảm biến IoT đóng vai trò nền tảng, bởi chúng đảm nhận nhiệm vụ thu thập dữ liệu từ môi trường thực tế và truyền về hệ thống xử lý trung tâm.

Thực tế hiện nay cho thấy nhu cầu về giám sát môi trường, quản lý năng lượng và điều khiển thiết bị điện tử từ xa ngày càng cao. Ví dụ, trong các hộ gia đình, việc kiểm soát nhiệt độ, độ ẩm hay ánh sáng có thể giúp tiết kiệm năng lượng, bảo vệ sức khỏe và nâng cao chất lượng cuộc sống. Trong sản xuất và công nghiệp, giám sát môi trường giúp kiểm soát chất lượng sản phẩm, đảm bảo an toàn lao động. Xuất phát từ nhu cầu đó, đề tài “*Xây dựng hệ thống cảm biến IoT*” được lựa chọn nhằm nghiên cứu, thiết kế và triển khai một mô hình IoT hoàn chỉnh có khả năng thu thập, lưu trữ, hiển thị và điều khiển dữ liệu cảm biến.

2. Làm gì?

Đề tài hướng đến việc xây dựng một hệ thống IoT có khả năng thực hiện các chức năng sau:

- Thu thập dữ liệu môi trường: Hệ thống sử dụng các cảm biến để đo các thông số cơ bản như nhiệt độ, độ ẩm và cường độ ánh sáng. Các dữ liệu này được thu nhận bởi vi điều khiển ESP32
- Truyền dữ liệu về máy chủ: Các dữ liệu sau khi được thu thập sẽ được gửi về máy chủ thông qua giao thức MQTT, đảm bảo tốc độ và độ tin cậy trong quá trình truyền tải.
- Lưu trữ dữ liệu: Dữ liệu được lưu trong cơ sở dữ liệu MySQL, giúp dễ dàng quản lý, truy vấn và phân tích.
- Hiển thị dữ liệu: Một giao diện web trực quan được xây dựng để hiển thị dữ liệu theo thời gian thực dưới dạng bảng và biểu đồ. Người dùng có thể dễ dàng theo dõi sự thay đổi của nhiệt độ, độ ẩm, ánh sáng theo từng khoảng thời gian.

3. Phát triển như thế nào?

Hệ thống được phát triển dựa trên kiến trúc ba lớp điển hình của IoT, bao gồm:

- Lớp thiết bị (Device Layer): Sử dụng vi điều khiển ESP32 kết nối với cảm biến DHT11 (đo nhiệt độ và độ ẩm) và BH1750 (đo ánh sáng). Các cảm biến thu thập dữ liệu và gửi về ESP32 để xử lý sơ bộ trước khi truyền đi.
- Lớp mạng (Network Layer): Dữ liệu được truyền qua mạng WiFi và giao thức MQTT thông qua broker (HiveMQ). Giao thức này

nhẹ, nhanh, phù hợp với các thiết bị IoT vốn hạn chế về tài nguyên.

- Lớp ứng dụng (Application Layer):
 - Backend: Xây dựng bằng Java Spring Boot, chịu trách nhiệm nhận dữ liệu từ MQTT, xử lý và lưu trữ vào MySQL. Backend cũng cung cấp các REST API và WebSocket để phục vụ frontend.
 - Frontend: Phát triển bằng HTML/CSS/JavaScript (hoặc ReactJS), sử dụng Chart.js để hiển thị biểu đồ dữ liệu cảm biến, đồng thời cung cấp giao diện trực quan cho người dùng điều khiển thiết bị.

Phương pháp phát triển của hệ thống được chia theo hướng phát triển từng chức năng. Quá trình triển khai bao gồm:

- Giai đoạn 1: Kết nối cảm biến và thu thập dữ liệu.
- Giai đoạn 2: Xây dựng backend để xử lý và lưu trữ dữ liệu.
- Giai đoạn 3: Phát triển frontend hiển thị dữ liệu theo thời gian thực.
- Giai đoạn 4: Kiểm thử hệ thống, đánh giá hiệu quả và tối ưu hóa.

4. Phạm vi nghiên cứu và triển khai

Do giới hạn về thời gian và nguồn lực, hệ thống được xây dựng ở quy mô thử nghiệm nhỏ, với một số lượng thiết bị giới hạn (2 thiết bị cảm biến: BHT11 và DH1750). Hệ thống tập trung vào các chức năng cốt lõi: thu thập, lưu trữ, hiển thị dữ liệu và điều khiển thiết bị. Các vấn đề nâng cao như bảo mật, mở rộng quy mô lớn, tích hợp trí tuệ nhân tạo để dự đoán xu hướng dữ

liệu chưa được triển khai trong phạm vi đề tài, nhưng có thể được nghiên cứu và bổ sung trong tương lai.

5. Ý nghĩa và đóng góp

Đề tài “*Xây dựng hệ thống cảm biến IoT*” có ý nghĩa quan trọng cả về mặt học thuật lẫn thực tiễn:

- Về mặt học thuật: Đề tài giúp sinh viên tiếp cận toàn diện quy trình phát triển một hệ thống IoT, từ việc lập trình nhúng, quản lý cơ sở dữ liệu, phát triển backend đến xây dựng giao diện người dùng. Đây là cơ hội rèn luyện kỹ năng tích hợp phần cứng và phần mềm, áp dụng các công nghệ hiện đại vào một sản phẩm cụ thể.
- Về mặt thực tiễn: Hệ thống có thể áp dụng trong việc giám sát môi trường trong hộ gia đình, lớp học, phòng thí nghiệm, hoặc mở rộng thành nền tảng cho các ứng dụng nhà thông minh và nông nghiệp thông minh.
- Về mặt định hướng tương lai: Đây là bước khởi đầu để nghiên cứu và phát triển các hệ thống IoT lớn hơn, tích hợp nhiều loại cảm biến khác nhau, hỗ trợ trí tuệ nhân tạo để phân tích dữ liệu và dự đoán xu hướng.

CHƯƠNG II: THIẾT KẾ HỆ THỐNG

1. Phân tích yêu cầu hệ thống

1.1. Phân tích yêu cầu hệ thống

- Thu thập dữ liệu cảm biến (nhiệt độ, độ ẩm, ánh sáng).
- Truyền dữ liệu về máy chủ qua giao thức MQTT.
- Lưu trữ dữ liệu vào cơ sở dữ liệu.
- Hiển thị dữ liệu theo thời gian thực (bảng, biểu đồ).

1.2. Yêu cầu phi chức năng

- Hệ thống phải hoạt động ổn định, liên tục.
- Dữ liệu được cập nhật theo thời gian thực.
- Khả năng mở rộng khi thêm nhiều cảm biến.
- Giao diện dễ sử dụng, trực quan.

2. Thiết kế phần cứng

2.1. ESP32 – DEVKIT

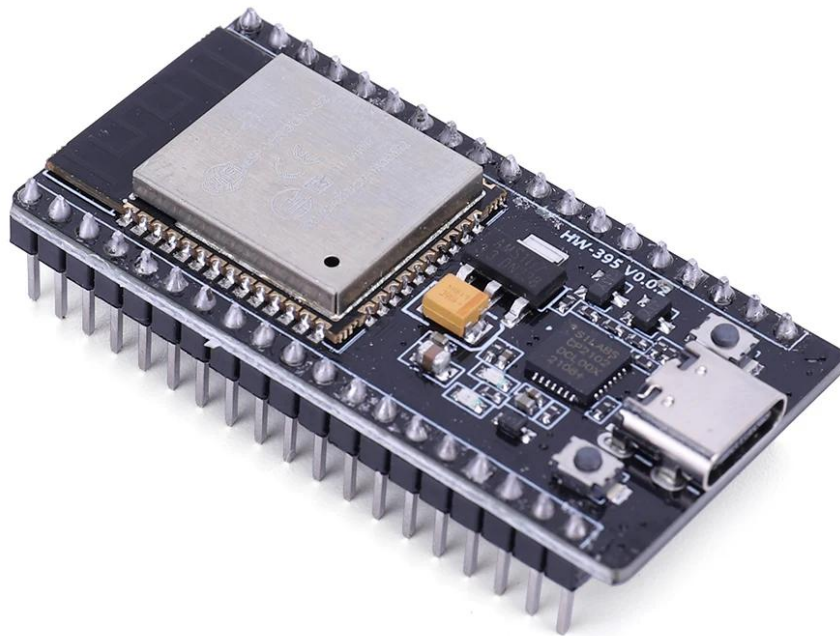
2.1.1. Giới thiệu

ESP32 là một bộ vi điều khiển hiệu suất cao thuộc dòng vi điều khiển tích hợp trên chip, nổi bật với khả năng tiêu thụ điện năng thấp và chi phí hợp lý. Đa số các biến thể của ESP32 đều tích hợp Wi-Fi và Bluetooth hai chế độ, mang lại tính linh hoạt, độ tin cậy cao và sức mạnh xử lý phù hợp với nhiều ứng dụng khác nhau.

Là thế hệ kế nhiệm của dòng NodeMCU ESP8266 phổ biến, ESP32 cung cấp hiệu năng vượt trội cùng nhiều tính năng nâng cao. Sản phẩm được

phát triển bởi **Espressif Systems** và được ứng dụng rộng rãi trong các lĩnh vực như Internet of Things (IoT), robot, và hệ thống tự động hóa.

Với thiết kế tối ưu cho các thiết bị sử dụng pin, ESP32 sở hữu hệ thống quản lý năng lượng thông minh, hỗ trợ các chế độ ngủ sâu (deep sleep), giúp kéo dài đáng kể thời gian sử dụng của thiết bị.



2.1.2. Tổng quan đặc điểm

- Cầu USB-to-UART: Tích hợp chip USB-to-UART đơn, hỗ trợ tốc độ truyền dữ liệu lên đến 3 Mbps.

- Giao diện kết nối: Sử dụng cổng USB Type-C, vừa để cấp nguồn, vừa làm giao tiếp dữ liệu giữa ESP32 và máy tính.
- Khả năng giao tiếp: Hỗ trợ kết nối Wi-Fi 2.4 GHz và Bluetooth v4.2 + BLE.
- Bộ xử lý trung tâm: Vi xử lý Xtensa LX6 lõi kép 32-bit, hiệu năng cao, hỗ trợ chế độ tiết kiệm điện năng.
- Thiết bị ngoại vi: Hỗ trợ nhiều ngoại vi với độ ổn định cao, tiêu thụ năng lượng tối ưu, phù hợp với nhiều yêu cầu ứng dụng thực tế.
- Phương thức cấp nguồn (3 tùy chọn):
 - Qua cổng micro-USB (mặc định)
 - Qua chân 5V / GND
 - Qua chân 3.3V / GND

2.1.3. Thông số kỹ thuật

Thành phần	Thông tin
Vi xử lý	Dual-core Tensilica Xtensa LX6, xung nhịp tối đa 240 MHz
RAM	520 KB SRAM
ROM	448 KB
Flash	4 MB (thường dùng)
Kết nối	Wi-Fi 802.11 b/g/n, Bluetooth v4.2 + BLE

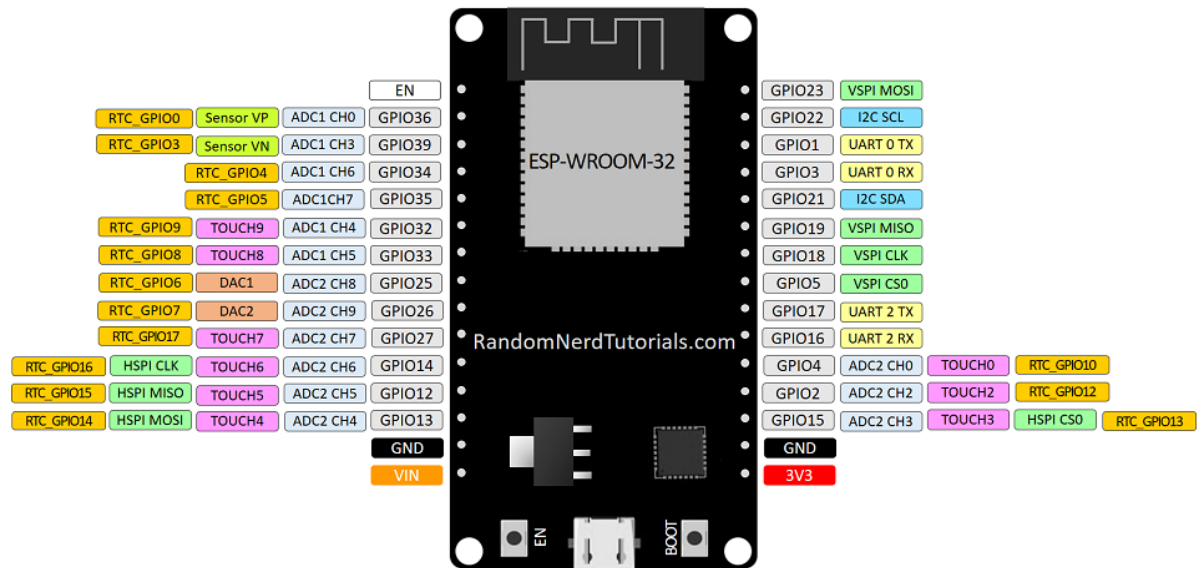
Nguồn cấp	<ul style="list-style-type: none"> - 5V qua cổng microUSB hoặc chân VIN - 3.3V qua chân 3V3
GPIO	30 chân, nhiều chân đa năng
ADC	12-bit, tối đa 18 kênh
DAC	2 kênh (GPIO25, GPIO26)
PWM	Hỗ trợ hầu hết các chân GPIO
UART/SPI/I2C/I2S	Hỗ trợ nhiều kênh, dễ cấu hình
Dòng tiêu thụ	<ul style="list-style-type: none"> - Bình thường: 160–260 mA - Chế độ ngủ sâu: $\sim 10 \mu\text{A}$
Kích thước	52 x 29 mm

2.1.4. Sơ đồ chân

DEVKIT hệ thống sử dụng có 30 chân kết nối, trong đó có 25 GPIO , 1 chân EN(rs), 2 chân GND và 2 chân nguồn

ESP32 DEVKIT V1 – DOIT

version with 30 GPIOs

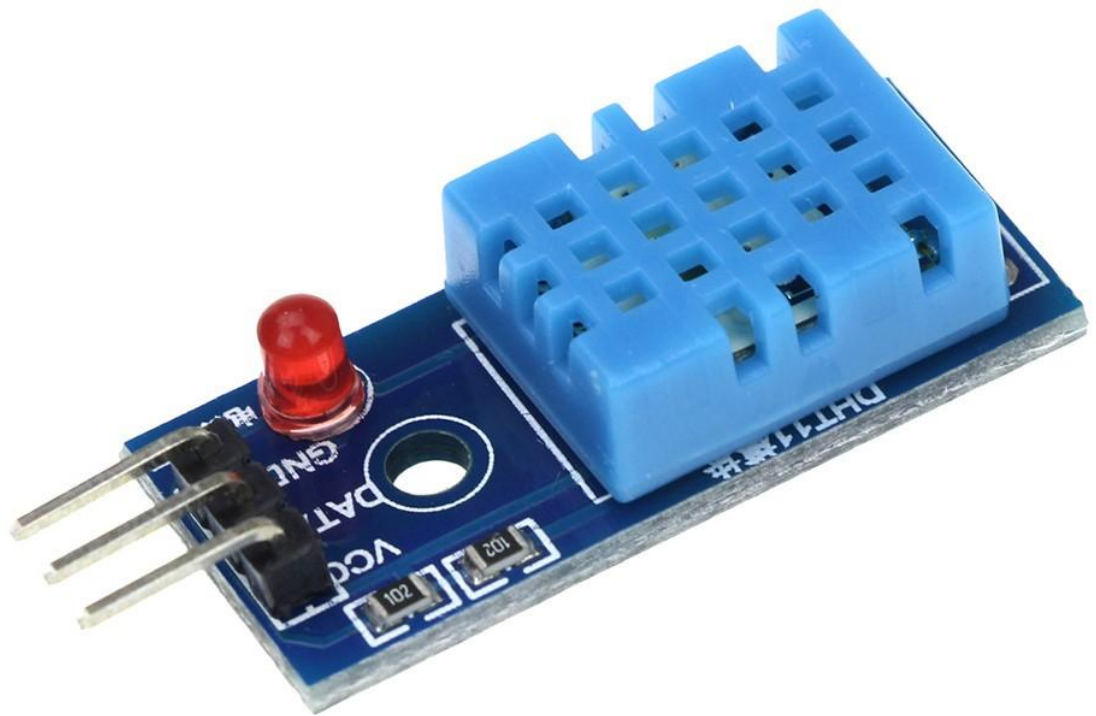


2.2. Module cảm biến nhiệt độ, độ ẩm DHT11

2.2.1. Giới thiệu

DHT11 là một cảm biến kỹ thuật số giá rẻ, chuyên dùng để đo nhiệt độ và độ ẩm không khí. Với khả năng giao tiếp dễ dàng thông qua giao thức 1-wire, cảm biến này có thể được sử dụng với nhiều loại vi điều khiển như Arduino, Raspberry Pi, ESP32,...

DHT11 sử dụng một điện trở nhiệt (thermistor) để đo nhiệt độ và một cảm biến độ ẩm điện dung để đo độ ẩm tương đối trong không khí. Đây là lựa chọn phổ biến trong các dự án IoT, điều khiển môi trường, hoặc thiết bị thời tiết nhúng nhờ chi phí thấp và độ tiện dụng.



2.2.2. Đặc điểm & Cấu tạo

- Cấu tạo cảm biến DHT11
 - Đo độ ẩm: Cảm biến sử dụng một tụ điện với hai điện cực và lớp vật liệu hấp thụ ẩm làm chất điện môi. Khi độ ẩm thay đổi, giá trị điện dung thay đổi theo, và IC nội bộ sẽ chuyển các thay đổi này thành tín hiệu số.

- Đo nhiệt độ: DHT11 sử dụng thermistor hệ số nhiệt độ âm (NTC), tức là điện trở của nó giảm khi nhiệt độ tăng. Các vật liệu thường dùng gồm gốm bán dẫn hoặc polymer, giúp tăng độ nhạy ngay cả với thay đổi nhiệt độ nhỏ.
- Tính năng nổi bật:
 - Giao tiếp 1-Wire: Truyền dữ liệu qua 1 chân duy nhất, đơn giản hóa kết nối.
 - Tích hợp bộ xử lý tín hiệu số: Trả về dữ liệu chính xác mà không cần xử lý phức tạp.
 - Giá thành rẻ, dễ tiếp cận cho người mới học và dự án quy mô nhỏ.
 - So với DHT22, DHT11 có độ chính xác và phạm vi đo hẹp hơn, nhưng vẫn đáp ứng tốt nhu cầu cơ bản.

2.2.3. Thông số kỹ thuật

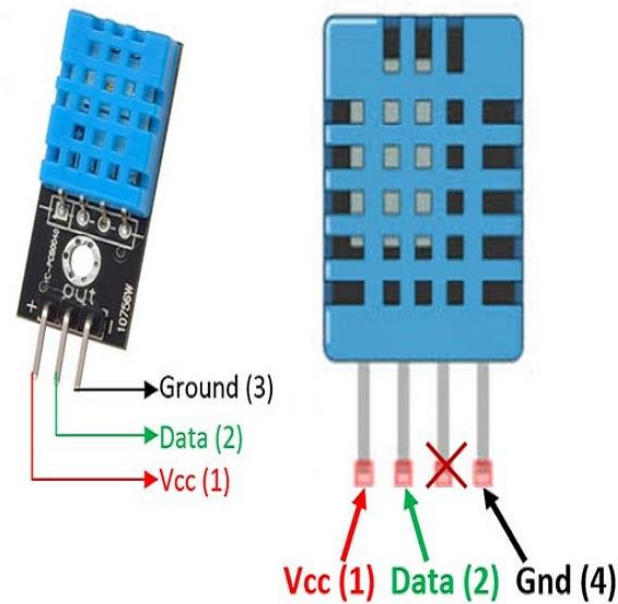
Thông số	Giá trị
Điện áp hoạt động	3V – 5V DC
Dòng tiêu thụ	2.5 mA
Phạm vi đo độ ẩm	20% – 90% RH
Độ chính xác độ ẩm	$\pm 5\%$ RH
Phạm vi đo nhiệt độ	0°C – 50°C
Độ chính xác nhiệt độ	$\pm 2^\circ\text{C}$

Tần số lấy mẫu	1Hz (1 lần/giây)
Giao tiếp	Digital 1-wire
Kích thước	23 x 12 x 5 mm

2.2.4. Sơ đồ chân DHT11

Cảm biến DHT11 tiêu chuẩn thường có 4 chân, nhưng trong các module phổ biến, chỉ 3 chân được sử dụng. Dưới đây là sơ đồ và chức năng từng chân:

Tên chân	Mô tả chức năng
VCC	Cấp nguồn cho cảm biến (3V – 5V DC)
DATA	Chân truyền dữ liệu (giao tiếp 1-wire)
NC	Không kết nối – thường bỏ trống
GND	Nối đất (Ground)

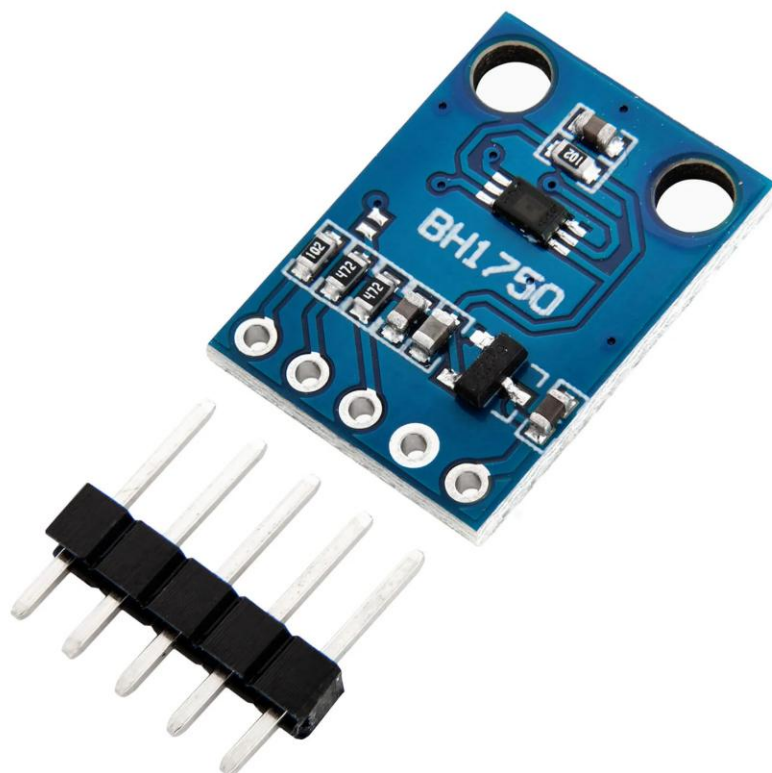


2.3. Module cảm biến ánh sáng BH1750

2.3.1. Giới thiệu

BH1750 là một cảm biến ánh sáng xung quanh kỹ thuật số, thường được tích hợp trong các thiết bị như điện thoại thông minh, thiết bị IoT, hoặc thiết bị điều khiển tự động, nhằm đo độ sáng của môi trường và điều chỉnh chức năng phù hợp.

BH1750 có khả năng đo cường độ ánh sáng (lux) với độ chính xác cao và dải đo rộng lên tới 65.535 lx. Cảm biến này giao tiếp qua giao thức I2C, giúp kết nối dễ dàng với các vi điều khiển như Arduino, ESP32, hoặc Raspberry Pi.



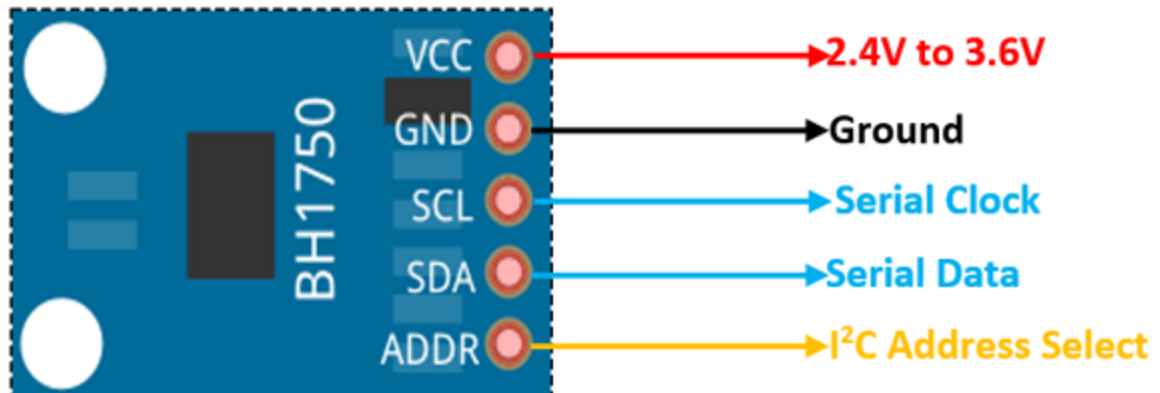
2.3.2. Thông số kỹ thuật

Thông số	Giá trị
Điện áp hoạt động	2.4V – 3.6V (thường sử dụng 3.0V)
Dòng điện tiêu thụ	Thấp hơn 0.12 mA
Dải đo ánh sáng	1 – 65.535 lux
Giao tiếp	I2C
Độ chính xác	±20%

Tích hợp A/D converter	Chuyển đổi tín hiệu analog ánh sáng thành dữ liệu số
Chống nhiễu hồng ngoại	Ảnh hưởng nhỏ bởi ánh sáng IR
Độ phản hồi	Gần giống với cảm nhận ánh sáng của mắt người

2.3.3. Sơ đồ chân

Tên chân	Mô tả chức năng
VCC	Cấp nguồn cho cảm biến (2.4V – 3.6V)
GND	Nối đất (Ground)
SCL	Chân xung đồng hồ (Clock line – I2C)
SDA	Chân dữ liệu (Data line – I2C)



2.4. Module Relay

2.4.1. Giới thiệu

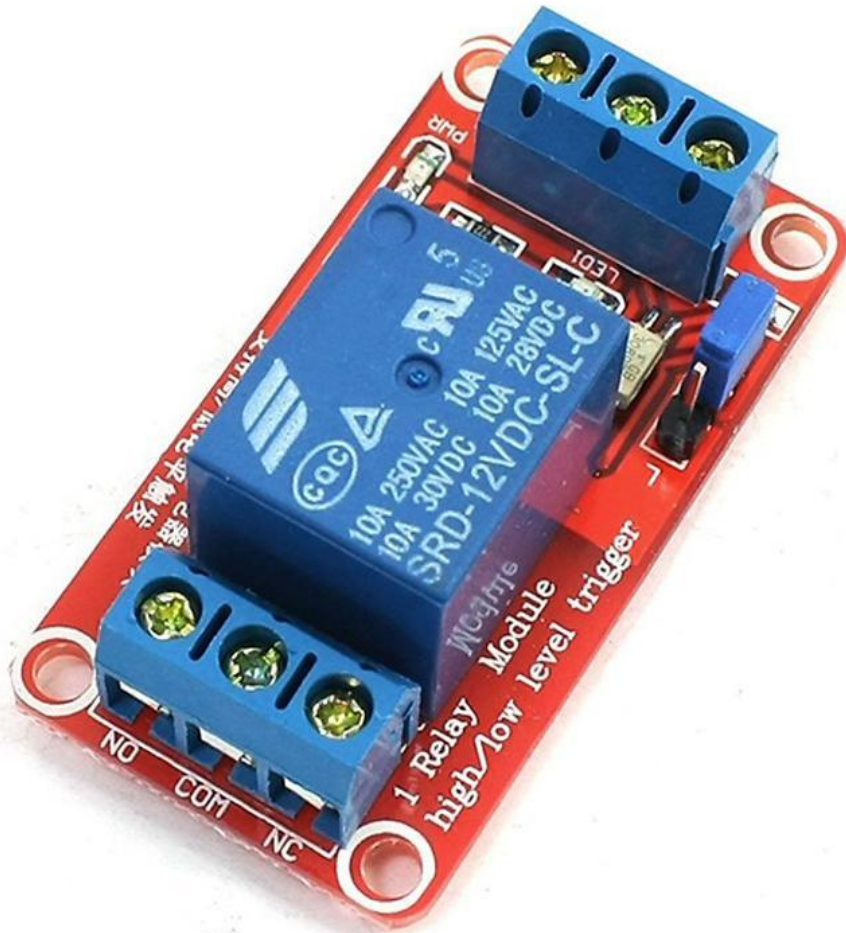
Relay là một linh kiện điện tử đóng vai trò như một công tắc điều khiển bằng điện. Khi nhận được tín hiệu điều khiển (thường là dòng điện nhỏ từ vi điều khiển như Arduino, ESP32...), relay sẽ kích hoạt để đóng hoặc ngắt một mạch điện công suất lớn hơn.

2.4.2. Cấu tạo

Module relay bao gồm nhiều thành phần quan trọng, mỗi bộ phận đảm nhận một chức năng riêng để đảm bảo hoạt động chính xác và an toàn của thiết bị:

- Cuộn dây (Coil): Là bộ phận tạo từ trường khi có dòng điện chạy qua, quyết định khả năng đóng/mở của relay.
- Tiếp điểm (Contacts): Thành phần quan trọng giúp relay đóng/mở mạch điện, điều khiển trực tiếp thiết bị.
- Diode bảo vệ: Giúp ngăn chặn dòng ngược từ cuộn dây, tránh làm hỏng vi điều khiển.

- Mạch cách ly (Optocoupler, Transistor): Đảm bảo sự an toàn và ổn định của tín hiệu điều khiển, giảm thiểu nhiễu điện.
- LED báo trạng thái: Hiển thị trạng thái hoạt động của relay, giúp người dùng dễ dàng kiểm tra.

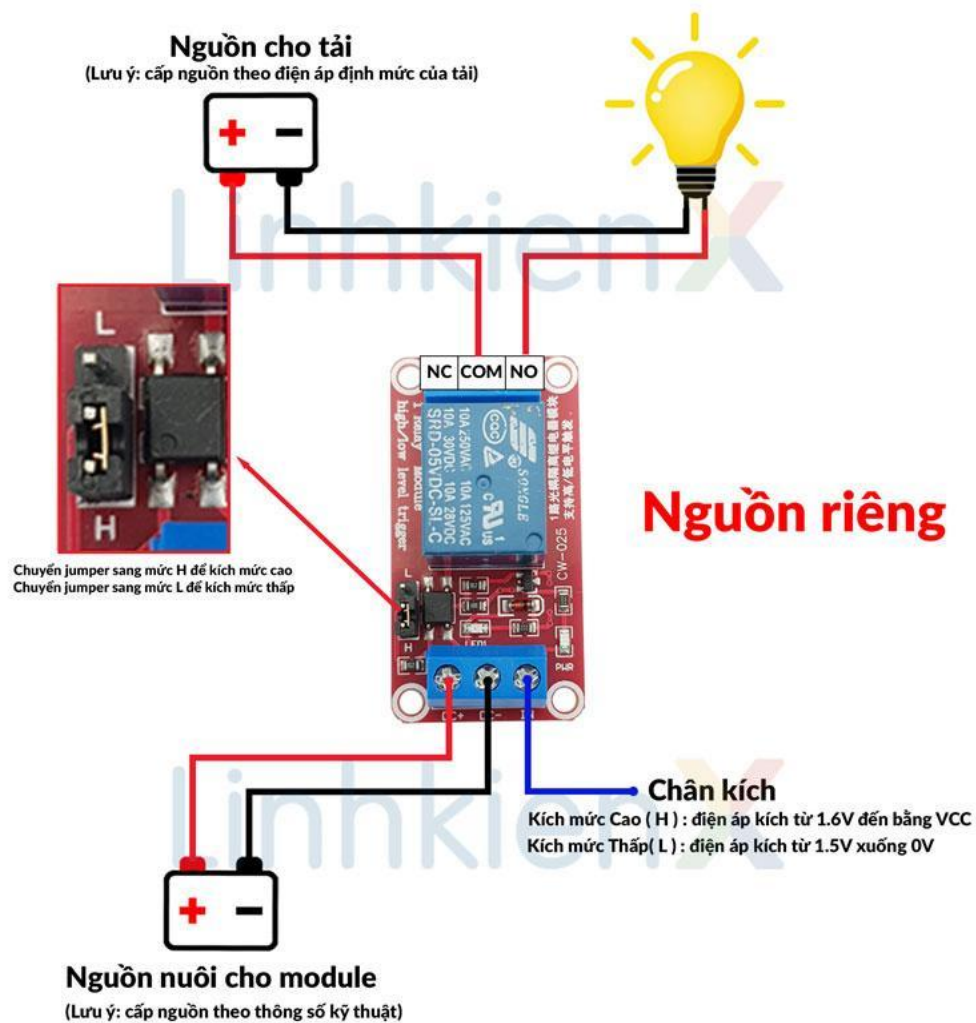


2.4.3. Sơ đồ hoạt động của Module Relay

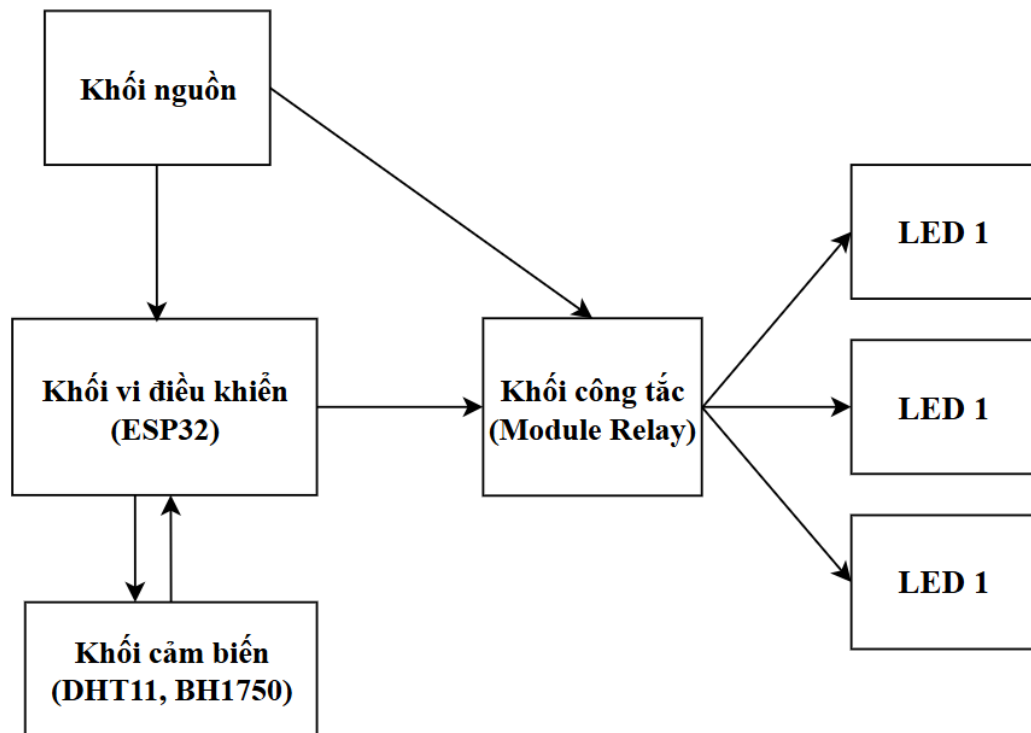
Khi vi điều khiển gửi tín hiệu điều khiển ở mức cao, dòng điện sẽ chạy qua cuộn dây bên trong relay, tạo ra từ trường làm hút tiếp điểm lại với nhau. Khi đó, tiếp điểm đóng mạch, cho phép dòng điện đi đến thiết bị tải,

giúp thiết bị bắt đầu hoạt động (ví dụ: đèn sáng, motor quay). Ngược lại, khi tín hiệu điều khiển chuyển về mức thấp, dòng điện qua cuộn dây bị ngắt, từ trường biến mất, và tiếp điểm trở lại trạng thái mở, khiến thiết bị ngừng hoạt động.

Tùy theo loại relay, tín hiệu điều khiển có thể là mức cao (active high) hoặc mức thấp (active low). Trong trường hợp relay kích mức thấp, tín hiệu điều khiển ở mức thấp mới kích hoạt cuộn dây hoạt động và đóng tiếp điểm.



2.5. Sơ đồ khối phần cứng



Khối nguồn:

- Cung cấp điện năng cho toàn bộ hệ thống, bao gồm:
 - Vi điều khiển ESP32
 - Các cảm biến (DHT11, BH1750)
 - Module relay
 - Thiết bị đầu ra (LED)
- Đảm bảo hệ thống hoạt động ổn định và liên tục.

Khối cảm biến (DHT11, BH1750)

- DHT11: Đo nhiệt độ và độ ẩm môi trường xung quanh.
- BH1750: Đo cường độ ánh sáng (lux).

- Các giá trị đo được sẽ được gửi về ESP32 để xử lý và ra quyết định điều khiển phù hợp.

- **Khối vi điều khiển (ESP32)**

- Là trung tâm điều khiển của hệ thống:
 - Thu thập dữ liệu từ các cảm biến.
 - Xử lý thông tin, phân tích điều kiện môi trường.
 - Gửi tín hiệu điều khiển đến khối relay để bật/tắt các tải.

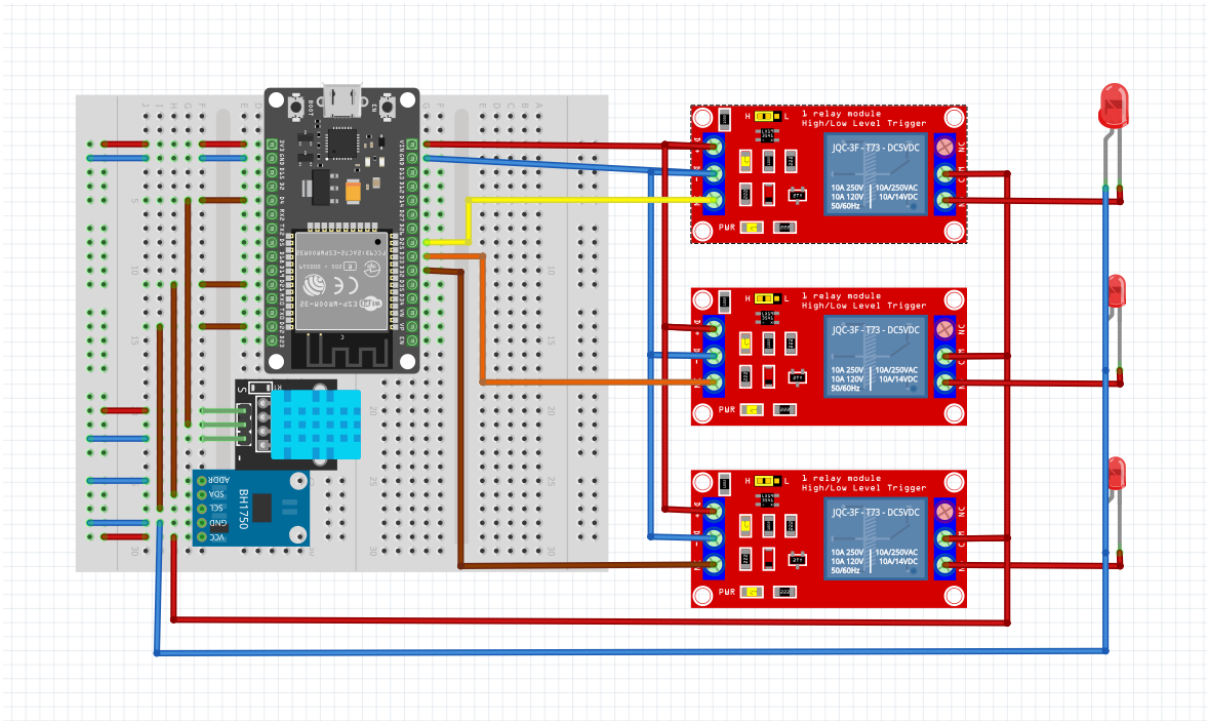
- **Khối công tắc điều khiển (Module Relay)**

- Đóng vai trò như một rơ-le điện tử:
 - Nhận tín hiệu điều khiển từ ESP32
 - Bật hoặc tắt các tải đầu ra (LED 1, LED 2, LED 3)

- **Các thiết bị đầu ra (LED 1, LED 2, LED 3)**

- Là các tải mô phỏng trong hệ thống.
- Được bật/tắt linh hoạt dựa trên điều kiện môi trường (ánh sáng, nhiệt độ, độ ẩm) hoặc lệnh từ người dùng.
- Điều khiển thông qua **module relay** do ESP32 kích hoạt.

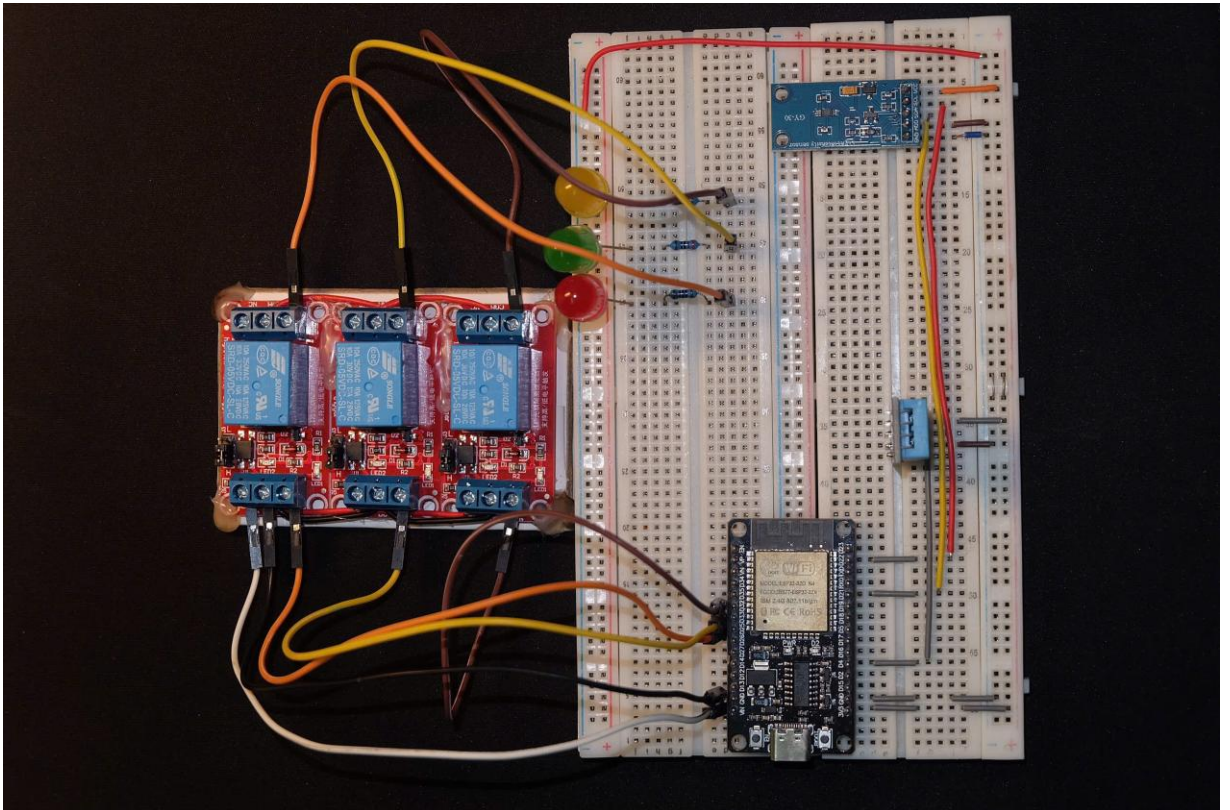
2.6. Sơ đồ đấu dây



- **Nguồn cấp cho hệ thống**
 - Toàn bộ mạch được cấp nguồn từ cổng USB của ESP32.
 - Điện áp 3.3V của ESP32 được dùng cho cảm biến.
 - Điện áp 5V được sử dụng để nuôi module relay và LED.
- **Khối vi điều khiển ESP32**
 - ESP32 được bố trí trên breadboard và đóng vai trò trung tâm xử lý.
 - Các chân GPIO của ESP32 được sử dụng để giao tiếp với cảm biến và phát tín hiệu điều khiển relay.
 - ESP32 được cấp nguồn trực tiếp từ cổng USB (5V), đồng thời sử dụng điện áp 3.3V nội bộ để nuôi cảm biến.
- **Khối cảm biến**

- **Cảm biến DHT11** dùng để đo nhiệt độ và độ ẩm.
 - Chân VCC của DHT11 nối với chân 3.3V trên ESP32.
 - Chân GND nối với GND của ESP32.
 - Chân DATA được kết nối với một chân GPIO của ESP32 để truyền dữ liệu.
- **Cảm biến BH1750** dùng để đo cường độ ánh sáng, giao tiếp với ESP32 qua giao thức I2C.
 - Chân VCC nối 3.3V.
 - Chân GND nối GND.
 - Chân SDA nối với GPIO21 (SDA) của ESP32.
 - Chân SCL nối với GPIO22 (SCL) của ESP32.
- **Khối điều khiển relay**
 - Hệ thống sử dụng 3 module relay đơn, mỗi relay điều khiển một LED riêng biệt.
 - Chân tín hiệu (IN) của mỗi relay lần lượt được kết nối đến các chân GPIO điều khiển trên ESP32.
 - Chân VCC của relay được cấp nguồn 5V, GND của relay nối chung mass với ESP32 để đồng bộ tín hiệu điều khiển.
- **Khối tải đầu ra (LED):**
 - Mỗi LED được nối với tiếp điểm **NO (Normally Open)** và **COM** của một relay.

- Khi ESP32 phát tín hiệu kích hoạt, relay đóng mạch cho phép dòng điện chạy qua LED và làm LED sáng.
- Nhờ vậy, vi điều khiển có thể điều khiển trạng thái bật/tắt của từng LED một cách độc lập.



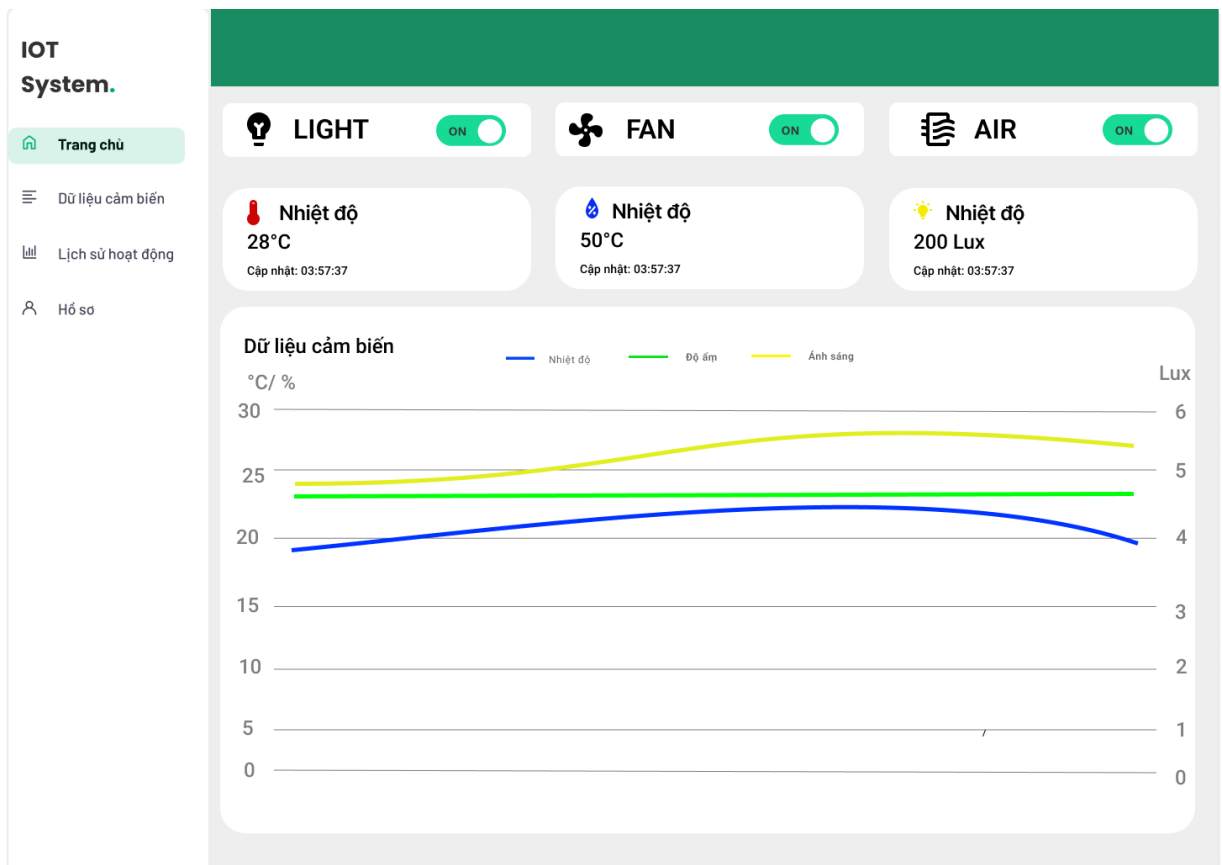
3. Thiết kế giao diện

Bộ cục tổng thể, thanh sidebar bên trái: hiển thị logo "IOT System." và các mục điều hướng:

- Trang chủ.
- Dữ liệu cảm biến.
- Lịch sử hoạt động.
- Hồ sơ.

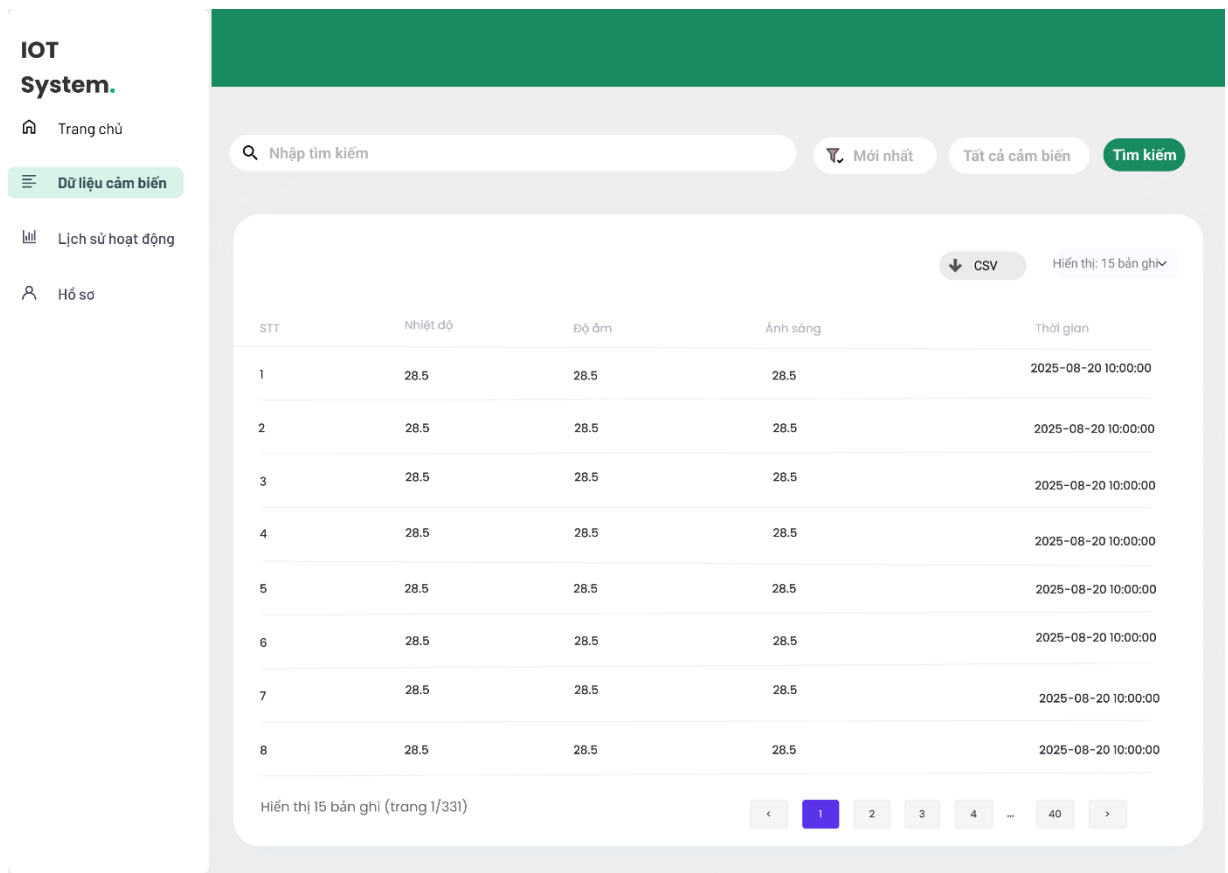
3.1. Giao diện trang chủ

- Khu vực điều khiển thiết bị, có 3 ô hiển thị trạng thái thiết bị:
 - LIGHT (bóng đèn): kèm icon bóng đèn, công tắc ON màu xanh.
 - FAN (quạt): kèm icon quạt gió, công tắc ON màu xanh.
 - AIR (điều hòa): kèm icon điều hòa, công tắc ON màu xanh.
- Khu vực cảm biến: Hiển thị thông số của cảm biến (nhiệt độ, độ ẩm, ánh sáng), giá trị cảm biến đo được và thời gian đo theo dạng HH:mm:ss
- Phân biểu đồ đường hiển thị 3 thông số: nhiệt độ, độ ẩm, ánh sáng. Trục tung bên trái hiển thị giá trị °C / %, trục tung bên phải hiển thị đơn vị Lux.



3.2. Giao diện dữ liệu cảm biến

- Thanh tìm kiếm và bộ lọc:
 - Ô nhập tìm kiếm: cho phép gõ từ khóa để tìm dữ liệu cảm biến theo tiêu chí cụ thể.
 - Nút lọc “Mới nhất/ Cũ nhất”: giúp hiển thị dữ liệu mới nhất/ cũ nhất được ghi nhận.
 - Nút lọc “Tất cả cảm biến/ Nhiệt độ / Độ ẩm/ Ánh sáng”: cho phép chọn loại cảm biến muốn hiển thị (ví dụ: nhiệt độ, độ ẩm, ánh sáng).
 - Nút “Tìm kiếm”: thực hiện truy vấn dữ liệu theo điều kiện đã nhập/chọn.
- Bảng dữ liệu cảm biến:
 - Cấu trúc bảng: số thứ tự(STT), nhiệt độ, độ ẩm, ánh sáng, thời gian (dd-MM-yyyy HH:mm:ss)
 - Các tính năng bảng: Xuất dữ liệu ra file CSV, tùy chọn số bản ghi hiển thị (ví dụ: 15 bản ghi/trang), phân trang.



3.3. Giao diện lịch sử hoạt động

- Trang “Lịch sử hoạt động” được thiết kế để người dùng theo dõi và tra cứu toàn bộ hành động điều khiển thiết bị trong hệ thống IoT, với các tính năng tìm kiếm, lọc và xuất dữ liệu tiện lợi
- Thanh tìm kiếm và bộ lọc:
 - Ô nhập tìm kiếm: hỗ trợ tìm kiếm lịch sử hoạt động, theo tên thiết bị.
 - Nút lọc “Mới nhất/ Cũ nhất”: giúp hiển thị dữ liệu mới nhất/ cũ nhất được ghi nhận.
 - Nút lọc “Tất cả thiết bị”: cho phép chọn loại thiết bị (ví dụ LIGHT, FAN, AIR).
 - Nút lọc “Tất cả hành động”: chọn loại hành động (ON, OFF).

- Nút “Tìm kiếm”: thực hiện truy vấn dữ liệu theo điều kiện đã nhập/chọn.
- Bảng lịch sử hoạt động:
 - Cấu trúc bảng: số thứ tự(STT), nhiệt độ, độ ẩm, ánh sáng, thời gian (dd-MM-yyyy HH:mm:ss)
 - Các tính năng bảng: Xuất dữ liệu ra file CSV, tùy chọn số bản ghi hiển thị (ví dụ: 15 bản ghi/trang), phân trang.


STT	Thiết bị	Hành động	Thời gian
1	LIGHT	ON	2025-08-20 10:00:00
2	FAN	OFF	2025-08-20 10:00:00
3	AIR	OFF	2025-08-20 10:00:00
4	LIGHT	OFF	2025-08-20 10:00:00
5	FAN	ON	2025-08-20 10:00:00
6	AIR	OFF	2025-08-20 10:00:00
7	FAN	ON	2025-08-20 10:00:00
8	AIR	ON	2025-08-20 10:00:00


3.4. Giao diện hồ sơ


- Trang “Hồ sơ” trong hệ thống được thiết kế với bố cục gọn gàng, thể hiện đầy đủ các thông tin nhận diện và dữ liệu liên quan đến người dùng.


- Bên dưới là các trường thông tin được bố trí theo dạng biểu mẫu, bao gồm: họ và tên, ngày sinh, mã sinh viên, mã lớp, liên kết GitHub, liên kết Figma, tài liệu PDF và tài liệu API.


IOT
System.

 Trang chủ

 Dữ liệu cảm biến

 Lịch sử hoạt động

 Hồ sơ



Bùi Ngọc Vũ
buingocvu@gmail.com

Họ và tên

Bùi Ngọc Vũ

Ngày sinh

Mã sinh viên

Mã lớp

Link github

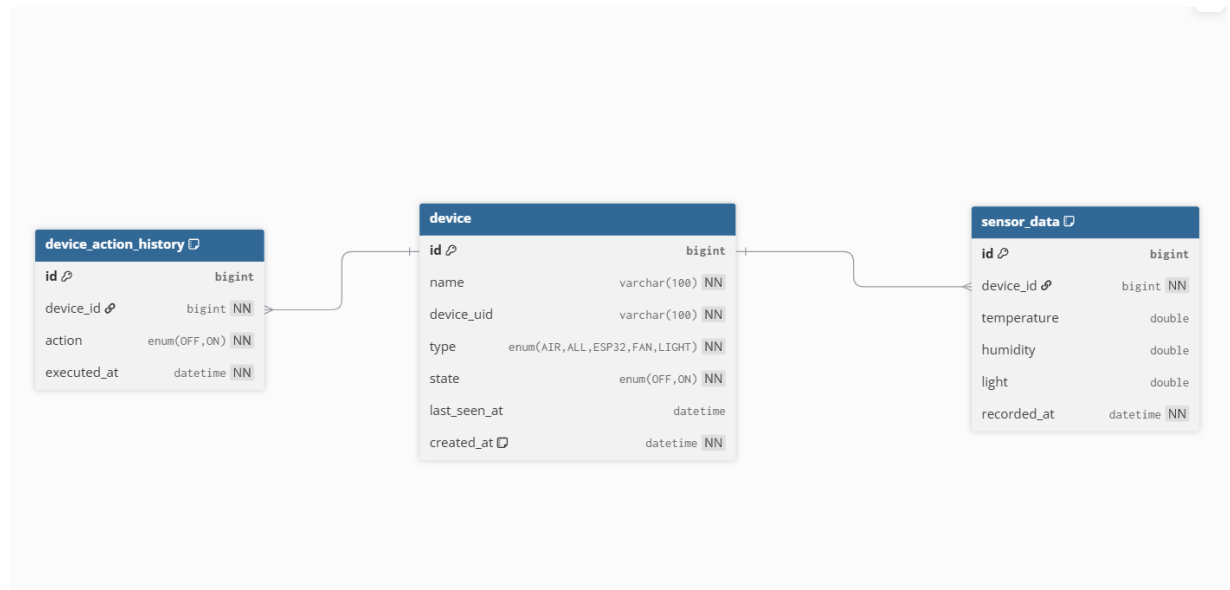
Link figma

PDF

API docs

4. Thiết kế cơ sở dữ liệu

4.1. Lược đồ ERD



4.2. Mối quan hệ giữa các bảng

4.2.1. Quan hệ giữa device và device_action_history:

- Bảng **device** là bảng trung tâm quản lý thông tin thiết bị IoT (đèn, quạt, máy lạnh, ESP32,...).
- Bảng **device_action_history** lưu lại lịch sử hành động bật/tắt của từng thiết bị.
- Mối quan hệ: 1–nhiều (one-to-many):
 - Mỗi thiết bị (**device**) có thể phát sinh nhiều hành động (**device_action_history**) trong quá trình vận hành.
 - Liên kết thông qua khóa ngoại **device_id** trong bảng **device_action_history** trỏ tới **id** của bảng **device**.

4.2.2. Quan hệ giữa device và sensor_data:

- Bảng sensor_data lưu trữ dữ liệu cảm biến đo được (nhiệt độ, độ ẩm, ánh sáng).
- Mỗi quan hệ: 1–nhiều (one-to-many):
 - Mỗi thiết bị (device) có thể phát sinh nhiều bản ghi dữ liệu (sensor_data) khi gửi dữ liệu đo đặc định kỳ.
 - Liên kết thông qua khóa ngoại device_id trong bảng sensor_data trở tới id của bảng device.

4.2.3. Tổng thể mối quan hệ

- device là bảng trung tâm, đóng vai trò định danh từng thiết bị trong hệ thống.
- device_action_history và sensor_data đều phụ thuộc vào device thông qua khóa ngoại.
- Sơ đồ quan hệ:
 - device → (1–n) → device_action_history
 - device → (1–n) → sensor_data

5. Sequence Diagram

5.1. Sequence 1: HomeFrm hiển thị thiết bị, dữ liệu cảm biến, điều khiển ON/OFF

5.1.1. Diễn giải

1. Người dùng truy cập vào màn HomeFrm
2. Hàm tạo HomeFrm() được gọi
3. HomeFrm() gọi phương thức getAll() của DeviceController qua API GET /api/devices

4. Phương thức `getAll()` của `DeviceController` được gọi
5. Phương thức `getAll()` gọi đến `DeviceService.getAllDevices()`
6. Phương thức `getAllDevices()` được gọi
7. `DeviceService` gọi truy cập entity để lấy dữ liệu bằng `deviceRepo.findAll()`
8. Entity `Device` gọi các phương thức getter (`getId()`, `getName()`, `getDeviceUid()`, `getType()`, `getState()`, `getLastSeenAt()`)
9. Entity `Device` trả danh sách đối tượng `Device` cho `DeviceRepository` rồi trả về cho `DeviceService`
10. `DeviceService` map sang DTO (`DeviceStatusDTO`) và trả kết quả cho `DeviceController`
11. `DeviceController` trả JSON danh sách thiết bị cho `HomeFrm()`
12. `HomeFrm` render danh sách thiết bị và đặt trạng thái ON/OFF cho từng công tắc
13. `HomeFrm()` gọi phương thức `getAll()` của `SensorDataController` qua API GET `/api/sensor-data` (hoặc gọi `search()` qua GET `/api/sensor-data/search` khi có tham số lọc)
14. Phương thức `getAll()` của `SensorDataController` được gọi
15. `SensorDataController` gọi `SensorDataService.getAllData()`
16. Phương thức `getAllData()` của `SensorDataController` được tạo
17. `getAllData()` lấy dữ liệu từ entity `SensorData` bằng phương thức `sensorRepo.search()`

18. Entity SensorData gọi các phương thức getter (getTemperature(), getHumidity(), getLight(), getRecordedAt())
19. Entity SensorData trả trang đối tượng cho SensorDataRepository và trả Page<SensorData> cho Service
20. SensorDataService map sang DTO (SensorReadingDTO) và gói trong PagedResponse
21. SensorDataController trả JSON dữ liệu cảm biến cho HomeFrm()
22. HomeFrm cập nhật giá trị nhiệt độ/độ ẩm/ánh sáng và đổ dữ liệu ban đầu vào biểu đồ
23. Người dùng thao tác công tắc ON/OFF trên HomeFrm.
24. HomeFrm gọi API POST /api/devices/command tới DeviceController với payload DeviceControlDTO (deviceId, action).
25. Phương thức sendCommand() của DeviceController được gọi.
26. DeviceController gọi DeviceService.sendCommand(dto).
27. Phương thức sendCommand(dto) được gọi.
28. DeviceService gọi CommandPublisher bằng phương thức deviceRepo.findById() để kiểm tra thiết bị và publish lệnh tới MQTT broker.
29. Phương thức sendAction() của CommandPublisher được gọi
30. Phương thức sendAction() của CommandPublisher gửi thông điệp topic MQTT với payload(deviceId, action, correlationId)
31. HomeFrm chuyển toggle sang "pending".

32. MQTT Broker chuyển lệnh đến ESP32.
33. ESP32 publish ACK lên "device_actions_ack" với payload JSON
34. CommandSubscriber nhận ACK
35. CommandSubscriber gọi DeviceStatusListener
36. Phương thức handleDeviceEvent() của DeviceStatusListener được gọi.
37. DeviceStatusListener tạo DeviceActionHistory mới và lưu qua ActionHistoryRepository
38. DeviceStatusListener gọi
wsTemplate.convertAndSend("/topic/devices", payload) để trả về HomeFrm
39. HomeFrm cập nhật trạng thái toggle và bỏ qua pending.
40. ESP32 publish dữ liệu lên MQTT topic "sensor/data" với payload JSON tới MQTT Broker
41. CommandSubscriber nhận message từ MQTT Broker
42. CommandSubscriber gọi tới CommandSubscriber với
topic="sensor/data" và message=<JSON>.
43. Phương thức handleDeviceEvent() parse JSON từ message và gọi
handleSensor() để xử lý:
 - Đọc deviceId, temperature, humidity, light/light_level từ payload
 - Xử lý giá trị (-1 được xem là sentinel → null)
 - Tìm Device theo deviceRepository.findById(deviceId)
44. sensorDataService.saveSensorData() được gọi để lưu dữ liệu vào DB.
45. Phương thức wsTemplate.convertAndSend("/topic/sensors",
wsPayload) được gọi
46. DeviceStatusListener đóng gói dữ liệu vào wsPayload để gửi realtime
tới HomeFrm
47. HomeFrm cập nhật box cảm biến, chartline
48. HomeFrm hiển thị cho người dùng.

```

sequenceDiagram
    participant User
    participant ESP32
    participant HomeFirm
    participant MQTTBroker as MQTT Broker
    participant wsTemplate
    participant DeviceController
    participant DeviceService
    participant SensorDataController
    participant SensorDataService
    participant CommandPublisher
    participant CommandSubscriber
    participant DeviceStatusListener
    participant ResponseUtils
    participant Device
    participant SensorData
    participant DB
    participant DeviceRepository
    participant SensorDataRepository
    participant ActorHistoryRepository

    Note over ESP32, HomeFirm: Load Device List
    ESP32->>HomeFirm: 1 Truy cập màn hình HomeFirm
    HomeFirm->>ESP32: 2 Tạo tạo HomeFirm()
    HomeFirm->>DeviceController: 3 GET /api/devices
    DeviceController->>DeviceService: 4 getApiDevices()
    DeviceService->>SensorDataController: 5 In(Info)
    SensorDataController->>SensorDataService: 6 Get getter (id, name, uid, type, state, lastSensor)
    SensorDataService->>DeviceRepository: 7 Trả danh sách Device
    DeviceRepository->>SensorDataRepository: 8 Trả danh sách Device
    SensorDataRepository->>ActorHistoryRepository: 9 Get getter (id, name, uid, type, state, lastSensor)
    ActorHistoryRepository->>SensorDataRepository: 10 Trả danh sách Device
    SensorDataRepository->>SensorDataService: 11 Trả DTO
    SensorDataService->>SensorDataController: 12 DTO danh sách thiết bị
    SensorDataController->>DeviceController: 13 Render danh sách + trạng thái ON/OFF
    DeviceController->>ESP32: 14 GET /api/sensor-data (thời gian hiện)
    DeviceController->>SensorDataController: 15 getApiData()
    SensorDataController->>SensorDataService: 16 search()
    SensorDataService->>DeviceRepository: 17 Get getter (temp, humi, light, recordDate)
    DeviceRepository->>SensorDataRepository: 18 Trả kết quả
    SensorDataRepository->>SensorDataService: 19 Page = SensorData
    SensorDataService->>SensorDataController: 20 Trả sang DTO (data = header DTO)
    SensorDataController->>DeviceController: 21 PageResponse
    DeviceController->>ESP32: 22 PageResponse
    ESP32->>DeviceController: 23 DTO dữ liệu cảm biến
    DeviceController->>ESP32: 24 hiển thị bảng + biểu đồ

    Note over ESP32, HomeFirm: User Toggle Device
    ESP32->>HomeFirm: 25 Thúc tác công tắc: ON/OFF
    HomeFirm->>DeviceController: 26 POST /api/devices/command (DeviceCommandDTO)
    DeviceController->>DeviceService: 27 sendCommandInfo()
    DeviceService->>SensorDataController: 28 findbyInfo()
    SensorDataController->>SensorDataService: 29 sendData()
    SensorDataService->>DeviceRepository: 30 Publish With (deviceId, action, correlationId)
    DeviceRepository->>SensorDataRepository: 31 Toggle chuyển sang 'pending'
    SensorDataRepository->>SensorDataService: 32 Gửi lên
    SensorDataService->>SensorDataController: 33 Publish ACK "device_@Info_ack"
    SensorDataController->>DeviceController: 34 Nhận ACK
    DeviceController->>DeviceService: 35 handleDeviceEvent()
    DeviceService->>SensorDataController: 36 saveDeviceCommandHistory()
    SensorDataController->>SensorDataService: 37 convertAndSend("topic/devices", payload)
    SensorDataService->>SensorDataController: 38 Update trạng thái toggle
    SensorDataController->>DeviceController: 39 Bỏ pending
    DeviceController->>ESP32: 40 Publish "sensor-data" (payload JSON)
    ESP32->>MQTTBroker: 41 Nhận message sensor-data
    MQTTBroker->>wsTemplate: 42 parse JSON
    wsTemplate->>DeviceController: 43 findbyDeviceId()
    DeviceController->>DeviceService: 44 saveSensorData()
    DeviceService->>SensorDataController: 45 Lưu dữ liệu
    SensorDataController->>DeviceController: 46 convertAndSend("topic/sensors", wsPayload)
    DeviceController->>ESP32: 47 Update box cảm biến + chartline
    ESP32->>HomeFirm: 48 hiển thị giao diện cập nhật
    HomeFirm->>DeviceController: 49 hiển thị giao diện cập nhật
  
```

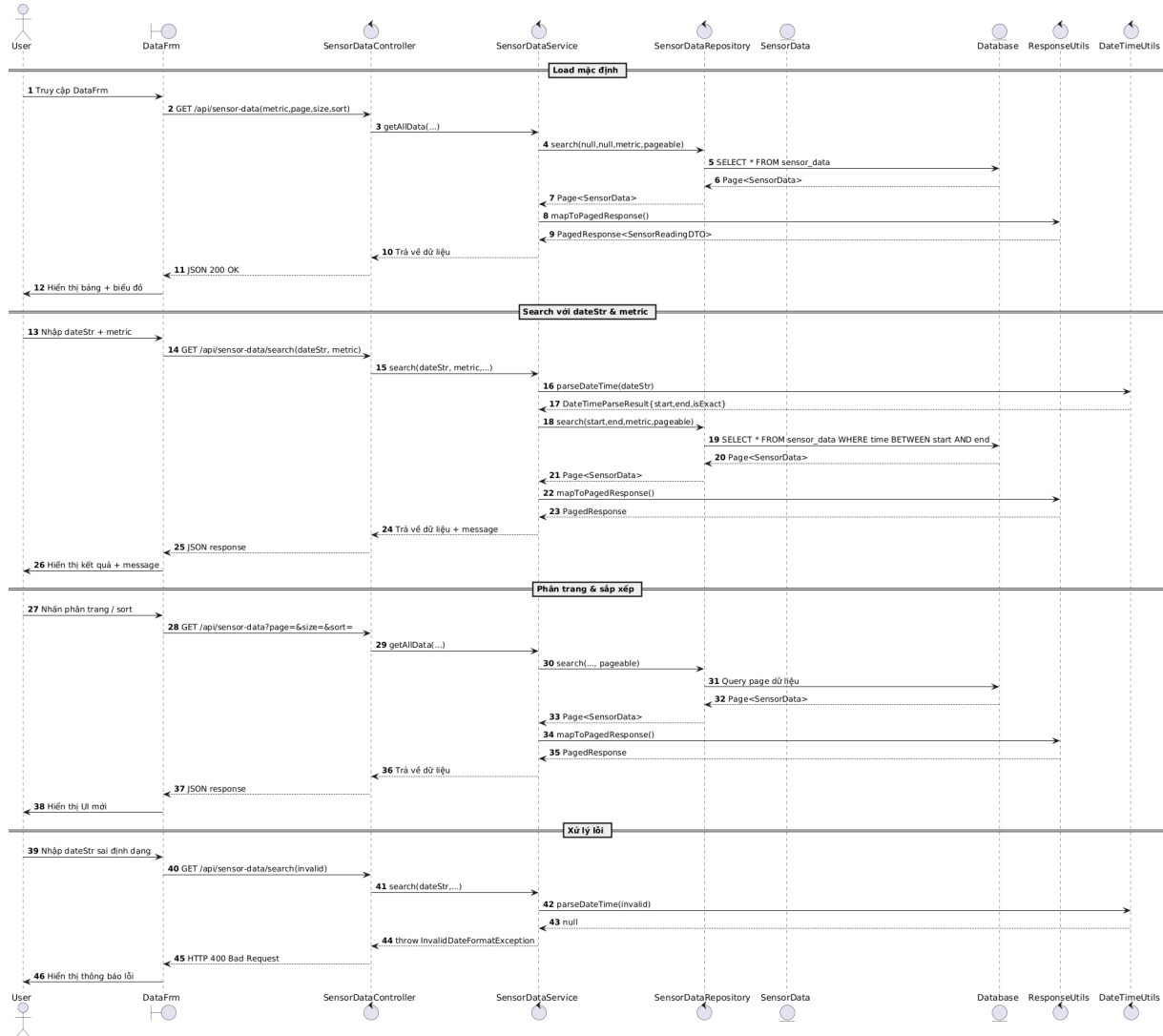
5.2.1. Diễn giải

1. Người dùng click menu "DỮ LIỆU CẢM BIẾN".
2. DataFrm (Boundary) được khởi tạo và hiển thị giao diện tìm kiếm mặc định.
3. DataFrm gọi API GET /api/sensor-data với tham số mặc định.
4. SensorDataController.getAll() (Control) nhận request.

5. SensorDataController gọi SensorDataService.getAllData(...).
6. SensorDataService tạo PageRequest.of(page, size, Sort.by("recordedAt")).
7. SensorDataService gọi SensorDataRepository.search(null, null, metric, pageable).
8. SensorDataRepository truy vấn Database.
9. Database trả về Page<SensorData>.
10. SensorDataRepository trả dữ liệu cho SensorDataService.
11. SensorDataService gọi
ResponseUtils.mapToPagedResponse(Page<SensorData>).
12. ResponseUtils trả PagedResponse<SensorReadingDTO> cho SensorDataService.
13. SensorDataService trả PagedResponse cho SensorDataController.
14. SensorDataController trả JSON (HTTP 200) cho DataFrm.
15. DataFrm hiển thị bảng và biểu đồ dữ liệu cảm biến.
16. Người dùng nhập dateStr và chọn metric → click "Tìm kiếm".
17. DataFrm gọi API GET /api/sensor-data/search?dateStr=...&metric=....
18. SensorDataController.search() nhận request và gọi
SensorDataService.search(...).
19. SensorDataService gọi DateTimeUtils.parseDateTime(dateStr).
20. DateTimeUtils trả DateTimeParseResult{start, end, isExact} hoặc ném lỗi.
21. Nếu hợp lệ, SensorDataService tạo PageRequest và gọi
SensorDataRepository.search(start,end,metric,pageable).
22. SensorDataRepository truy vấn Database.
23. Database trả về Page<SensorData>.
24. SensorDataRepository trả dữ liệu cho SensorDataService.
25. SensorDataService ánh xạ sang
PagedResponse<SensorReadingDTO> qua ResponseUtils.
26. SensorDataService trả PagedResponse cho SensorDataController.
27. SensorDataController trả JSON cho DataFrm.
28. DataFrm cập nhật bảng, biểu đồ và message kết quả.
29. Người dùng thay đổi page/size hoặc chọn metric khác.
30. DataFrm gửi request GET /api/sensor-data?page=&size=&metric=....

31. SensorDataController nhận request và gọi SensorDataService.getAllData(...).
32. SensorDataService tạo PageRequest mới và gọi SensorDataRepository.search(...).
33. SensorDataRepository truy vấn Database.
34. Database trả về Page<SensorData>.
35. SensorDataRepository trả dữ liệu cho SensorDataService.
36. SensorDataService ánh xạ qua ResponseUtils sang PagedResponse<SensorReadingDTO>.
37. SensorDataService trả PagedResponse cho SensorDataController.
38. SensorDataController trả JSON cho DataFrm.
39. DataFrm hiển thị dữ liệu mới (bảng/biểu đồ).
40. Người dùng click tiêu đề cột để sắp xếp.
41. DataFrm gửi request với tham số sort=asc/desc.
42. SensorDataService tạo Sort.by("recordedAt").ascending()/descending().
43. SensorDataRepository thực hiện query với sort mới.
44. Database trả về kết quả.
45. Kết quả đi qua SensorDataRepository → SensorDataService → SensorDataController.
46. DataFrm cập nhật bảng/biểu đồ theo dữ liệu đã sắp xếp.

5.2.2. Sequence



CHƯƠNG III. XÂY DỰNG HỆ THỐNG

1. Arduino

1.1. Thiết lập và triển khai thư viện

```
1  #include <Wire.h>
2  #include <BH1750.h>
3  #include "DHT.h"
4  #include <WiFi.h>
5  #include <WiFiClientSecure.h>
6  #include <PubSubClient.h>
7  #include <ArduinoJson.h>
8
```

- Wire.h: Thư viện giao tiếp I2C để kết nối với cảm biến BH1750
- BH1750.h: Thư viện điều khiển cảm biến ánh sáng BH1750
- DHT.h: Thư viện điều khiển cảm biến nhiệt độ và độ ẩm DHT11
- WiFi.h và WiFiClientSecure.h: Cho kết nối WiFi với bảo mật SSL/TLS
- PubSubClient.h: Hỗ trợ giao thức MQTT
- ArduinoJson.h: Xử lý dữ liệu định dạng JSON

1.2. Cấu hình hệ thống

```
9  #define DHTPIN 4
10 #define DHTTYPE DHT11
11 #define RELAY_TEMP 25
12 #define RELAY_LIGHT 33
13 #define RELAY_HUMI 32
14
```

- DHTPIN: Chân kết nối với cảm biến DHT11 (GPIO4)
- RELAY_TEMP/LIGHT/HUMI: Chân điều khiển các relay cho nhiệt độ, ánh sáng và độ ẩm

```

14
15 const char* ssid = "free wifi";
16 const char* password = "tu1den10";
17
18 const char* mqtt_server = "7c3aa4ee26624b92a6748300e938cd6b.s1.eu.hivemq.cloud";
19 const int    mqtt_port    = 8883;
20 const char* mqtt_user    = "esp32";
21 const char* mqtt_pass    = "Bnv2003@";
22

```

- Cấu hình kết nối WiFi và MQTT
- Sử dụng HiveMQ Cloud làm MQTT broker với kết nối bảo mật (cổng 8883)

```

23 const char* TOPIC_ACTIONS      = "device_actions";
24 const char* TOPIC_ACTIONS_ACK = "device_actions_ack";
25 const char* TOPIC_SENSOR      = "sensor/data";
26 const char* TOPIC_WILL        = "device/status";
27 const char* DEVICE_UID        = "esp32-1";
28

```

- Các topic MQTT được sử dụng trong hệ thống:
 - TOPIC_ACTIONS: Nhận lệnh điều khiển thiết bị
 - TOPIC_ACTIONS_ACK: Phản hồi sau khi thực hiện lệnh
 - TOPIC_SENSOR: Gửi dữ liệu cảm biến
 - TOPIC_WILL: Thông báo trạng thái thiết bị (online/offline)

```

28
29 WiFiClientSecure espClient;
30 PubSubClient client(espClient);
31
32 DHT dht(DHTPIN, DHTTYPE);
33 BH1750 lightMeter;
34

```

- espClient: Đối tượng WiFi client bảo mật
- client: Đối tượng MQTT client
- dht: Đối tượng điều khiển cảm biến DHT11

- lightMeter: Đối tượng điều khiển cảm biến ánh sáng BH1750

```

35  bool overrideTemp = false;
36  bool overrideHumi = false;
37  bool overrideLight = false;
38  unsigned long overrideTempAt = 0;
39  unsigned long overrideHumiAt = 0;
40  unsigned long overrideLightAt = 0;
41  const unsigned long overrideTimeoutMs = 60000;
42
43  unsigned long lastSampleAt = 0;
44  const unsigned long sampleEveryMs = 3000;
45
34
35  bool overrideTemp = false;
36  bool overrideHumi = false;
37  bool overrideLight = false;
38  unsigned long overrideTempAt = 0;
39  unsigned long overrideHumiAt = 0;
40  unsigned long overrideLightAt = 0;
41  const unsigned long overrideTimeoutMs = 60000;
42

```

- Các biến để theo dõi trạng thái điều khiển thủ công (override)
- Sau 60 giây (60000ms), hệ thống sẽ tự động trở về chế độ tự động
- Quản lý thời gian lấy mẫu dữ liệu cảm biến (mỗi 3 giây)

```

45
46  void connectWiFi() {
47      WiFi.mode(WIFI_STA);
48      WiFi.setSleep(false);
49      WiFi.begin(ssid, password);
50      while (WiFi.status() != WL_CONNECTED) { delay(200); }
51  }
52

```

- Thiết lập ESP32 ở chế độ station (kết nối đến router)
- Tắt chế độ sleep để duy trì kết nối liên tục
- Chờ đến khi kết nối WiFi thành công

```

52
53 void onMessage(char* topic, byte* payload, unsigned int length) {
54     StaticJsonDocument<256> doc;
55     if (deserializeJson(doc, payload, length)) return;
56
57     int deviceId = doc["deviceId"] | 0;
58     const char* action = doc["action"] | "";
59     const char* cid = doc["correlationId"] | "";
60     bool turnOn = strcmp(action, "ON") == 0;
61
62     if (deviceId == 1) { digitalWrite(RELAY_TEMP, turnOn ? HIGH : LOW); overrideTemp = true; overrideTempAt = millis(); }
63     if (deviceId == 2) { digitalWrite(RELAY_LIGHT, turnOn ? HIGH : LOW); overrideLight = true; overrideLightAt = millis(); }
64     if (deviceId == 3) { digitalWrite(RELAY_HUMI, turnOn ? HIGH : LOW); overrideHumi = true; overrideHumiAt = millis(); }
65
66     StaticJsonDocument<200> ack;
67     ack["deviceId"] = deviceId;
68     ack["state"] = action;
69     ack["correlationId"] = cid;
70     char buf[200];
71     size_t n = serializeJson(ack, buf, sizeof(buf));
72     client.publish(TOPIC_ACTIONS_ACK, (uint8_t*)buf, n, false);
73 }
74

```

- Xử lý tin nhắn điều khiển nhận từ server
- Parse JSON để lấy thông tin deviceId, action và correlationId
- Điều khiển relay tương ứng dựa trên deviceId và action
- Đánh dấu trạng thái override và thời điểm bắt đầu override
- Gửi phản hồi (acknowledgement) về server kèm trạng thái hiện tại

```

74
75 void resubscribe() {
76     client.subscribe(TOPIC_ACTIONS);
77 }
78
79 void reconnectMQTT() {
80     String clientId = "ESP32-" + String((uint32_t)ESP.getEfuseMac(), HEX);
81     while (!client.connected()) {
82         client.connect(clientId.c_str(), mqtt_user, mqtt_pass, TOPIC_WILL, 1, true, "offline");
83         if (client.connected()) {
84             client.publish(TOPIC_WILL, "online", true);
85             resubscribe();
86         } else {
87             delay(1000);
88         }
89     }
90 }
91

```

- resubscribe(): Đăng ký lại các topic sau khi kết nối lại MQTT
- reconnectMQTT(): Xử lý kết nối MQTT với:
 - ClientID duy nhất dựa trên địa chỉ MAC của ESP32

- Thiết lập Last Will Testament (LWT) để thông báo khi thiết bị mất kết nối
- Thông báo "online" khi kết nối thành công
- Đăng ký lại các topic cần thiết

```

91
92 void autoRelay(float t, float h, float lux) {
93     if (overrideTemp && millis() - overrideTempAt > overrideTimeoutMs) overrideTemp = false;
94     if (overrideHumi && millis() - overrideHumiAt > overrideTimeoutMs) overrideHumi = false;
95     if (overrideLight && millis() - overrideLightAt > overrideTimeoutMs) overrideLight = false;
96
97     if (!overrideTemp) digitalWrite(RELAY_TEMP, (!isnan(t) && t < 25) ? HIGH : LOW);
98     if (!overrideHumi) digitalWrite(RELAY_HUMI, (!isnan(h) && h < 10) ? HIGH : LOW);
99     if (!overrideLight) digitalWrite(RELAY_LIGHT, (lux > 0 && lux < 20) ? HIGH : LOW);
100 }
101

```

- Kiểm tra thời gian override đã vượt quá thời gian tối đa (60 giây)
- Tự động điều khiển relay dựa trên dữ liệu cảm biến nếu không trong chế độ override:
 - Relay nhiệt độ bật khi nhiệt độ dưới 25°C
 - Relay độ ẩm bật khi độ ẩm dưới 10%
 - Relay ánh sáng bật khi ánh sáng dưới 20 lux

```

102 void setup() {
103     Serial.begin(115200);
104     pinMode(RELAY_TEMP, OUTPUT);
105     pinMode(RELAY_LIGHT, OUTPUT);
106     pinMode(RELAY_HUMI, OUTPUT);
107     digitalWrite(RELAY_TEMP, LOW);
108     digitalWrite(RELAY_LIGHT, LOW);
109     digitalWrite(RELAY_HUMI, LOW);
110
111     connectWifi();
112     Wire.begin(21, 22);
113     lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE);
114     dht.begin();
115
116     espClient.setInsecure();
117     client.setServer(mqtt_server, mqtt_port);
118     client.setCallback(onMessage);
119     client.setKeepAlive(30);
120     client.setSocketTimeout(60);
121     client.setBufferSize(512);
122 }
123

```

- Khởi tạo serial monitor với baudrate 115200

- Thiết lập chân GPIO cho các relay và đặt trạng thái ban đầu là OFF (LOW)
- Kết nối WiFi
- Khởi tạo I2C (SDA=21, SCL=22) và các cảm biến
- Cấu hình MQTT client:
 - Sử dụng chế độ không xác minh chứng chỉ SSL (setInsecure())
 - Đặt server và callback function
 - Cấu hình keep-alive, timeout và buffer size

```

124 void loop() {
125   if (WiFi.status() != WL_CONNECTED) connectWiFi();
126   if (!client.connected()) reconnectMQTT();
127   client.loop();
128
129   unsigned long now = millis();
130   if (now - lastSampleAt >= sampleEveryMs) {
131     lastSampleAt = now;
132
133     float lux = lightMeter.readLightLevel();
134     float h = dht.readHumidity();
135     float t = dht.readTemperature();
136
137     StaticJsonDocument<256> doc;
138     doc["deviceId"] = DEVICE_UID;
139     if (!isnan(t)) doc["temperature"] = t; else doc["temperature"] = nullptr;
140     if (!isnan(h)) doc["humidity"] = h; else doc["humidity"] = nullptr;
141     if (!isnan(lux)) doc["light"] = lux; else doc["light"] = nullptr;
142
143     char payload[256];
144     size_t n = serializeJson(doc, payload, sizeof(payload));
145     client.publish(TOPIC_SENSOR, (uint8_t*)payload, n, false);
146
147     autoRelay(t, h, lux);
148   }
149 }

```

- Đảm bảo kết nối WiFi và MQTT luôn duy trì
- Xử lý sự kiện MQTT thông qua client.loop()
- Định kỳ mỗi 3 giây:
 - Đọc dữ liệu từ các cảm biến
 - Tạo JSON chứa dữ liệu và ID thiết bị

- Kiểm tra giá trị cảm biến hợp lệ trước khi gửi
- Gửi dữ liệu đến MQTT broker
- Cập nhật trạng thái relay dựa trên dữ liệu cảm biến

2. Cấu trúc hệ thống.

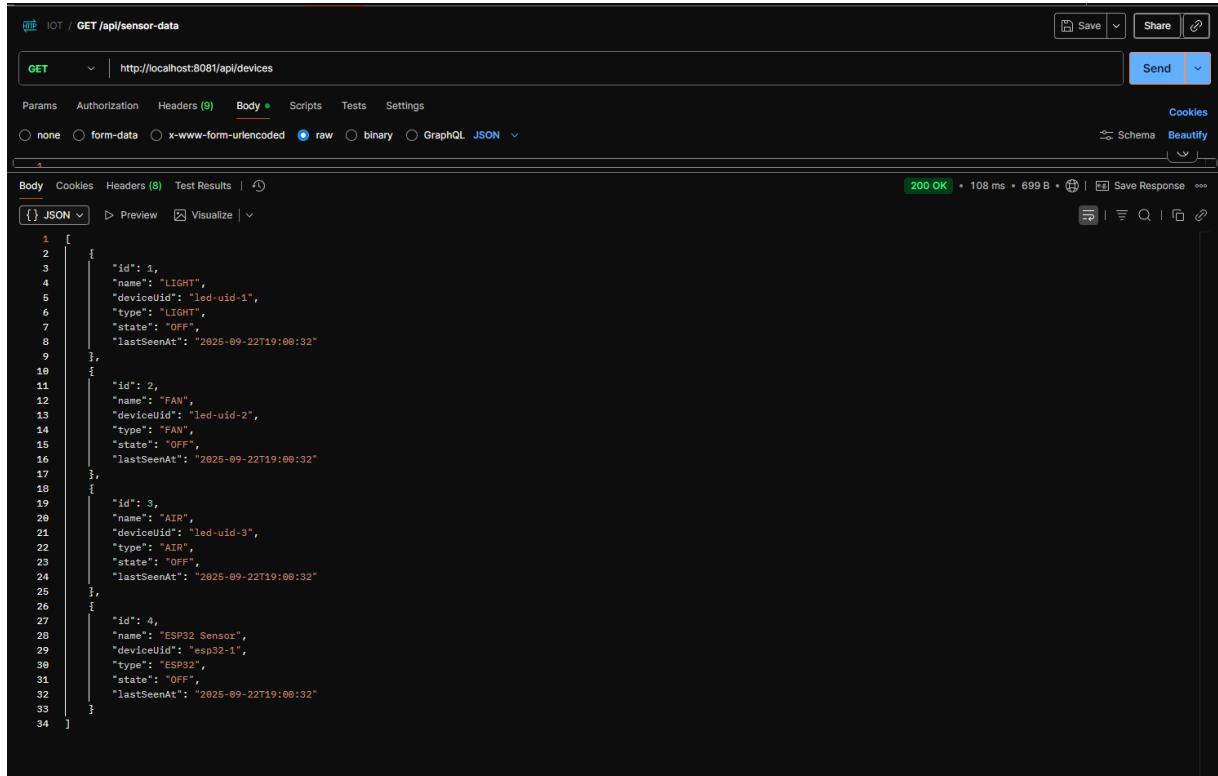
```

iot-system/
├── src/main/java/com/iot_system/
│   ├── config/                # Cấu hình (CORS, MQTT, WebSocket)
│   ├── controller/           # REST Controllers
│   ├── domain/
│   │   ├── dto/              # Data Transfer Objects
│   │   ├── entity/           # JPA Entities
│   │   └── enums/            # Enumerations
│   ├── exception/            # Global Exception Handler
│   ├── mqtt/                 # MQTT Publisher/Subscriber
│   ├── repository/           # JPA Repositories
│   ├── service/              # Business Logic
│   └── util/                  # Utility Classes
├── src/main/resources/
│   ├── static/               # Frontend files
│   │   ├── css/              # Stylesheets
│   │   ├── js/               # JavaScript files
│   │   ├── img/              # Images
│   │   └── *.html            # HTML pages
│   └── application.properties # Application config
├── target/                   # Maven build output
├── pom.xml                   # Maven dependencies
├── .env.example              # Environment variables template
├── ENV_SETUP.md              # Environment setup guide
├── README.md                 # This file
└── ...

```

3. Test API

3.1. Lấy danh sách thiết bị



3.2. Gửi lệnh điều khiển

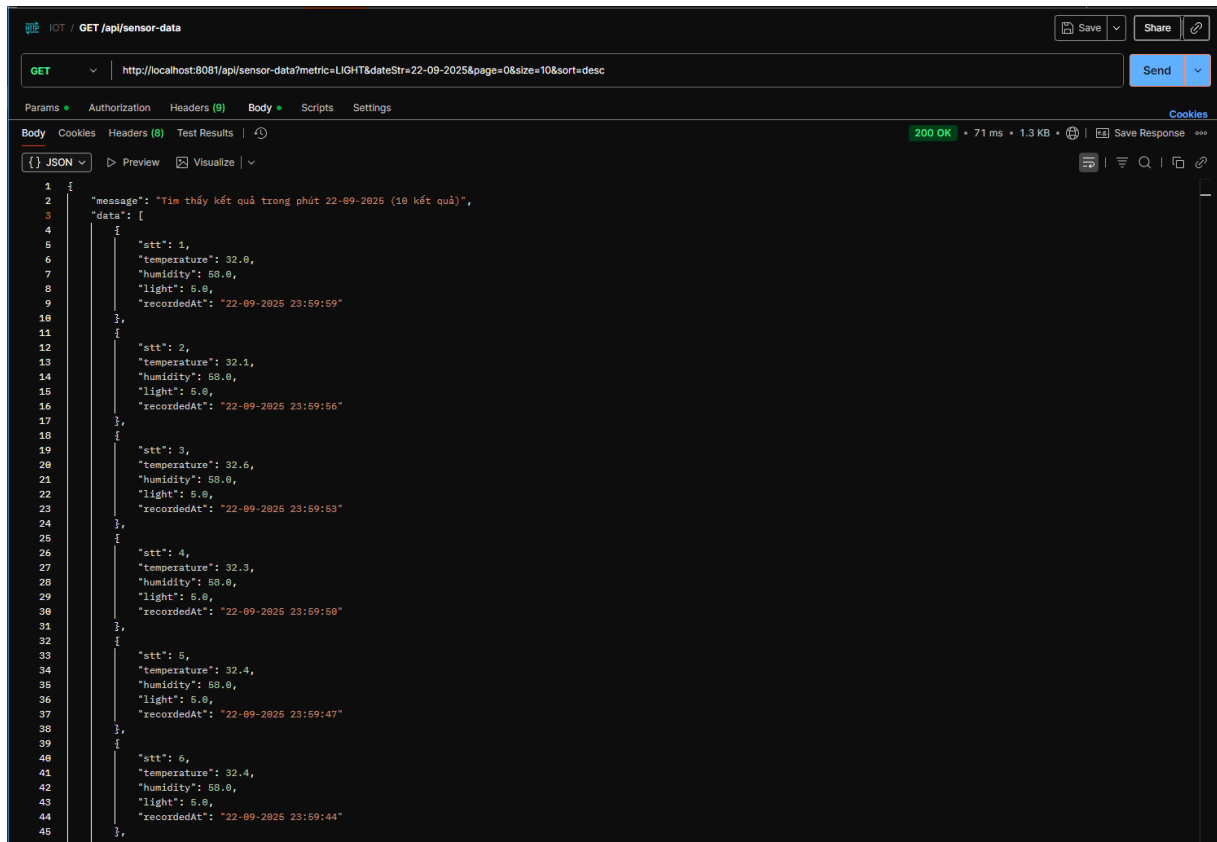
The screenshot displays a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8081/api/devices/command
- Body Type:** raw
- Request Body:**

```
1 {  
2   "deviceId": 1,  
3   "action": "OFF"  
4 }  
5
```
- Response Status:** 202 Accepted
- Response Body (JSON):**

```
1 {  
2   "status": "ACCEPTED",  
3   "correlationId": "9a38b666-c332-4a75-9e46-f7f1206c484a"  
4 }
```

3.3. Tìm kiếm có điều kiện



POSTMAN IOT / GET /api/sensor-data

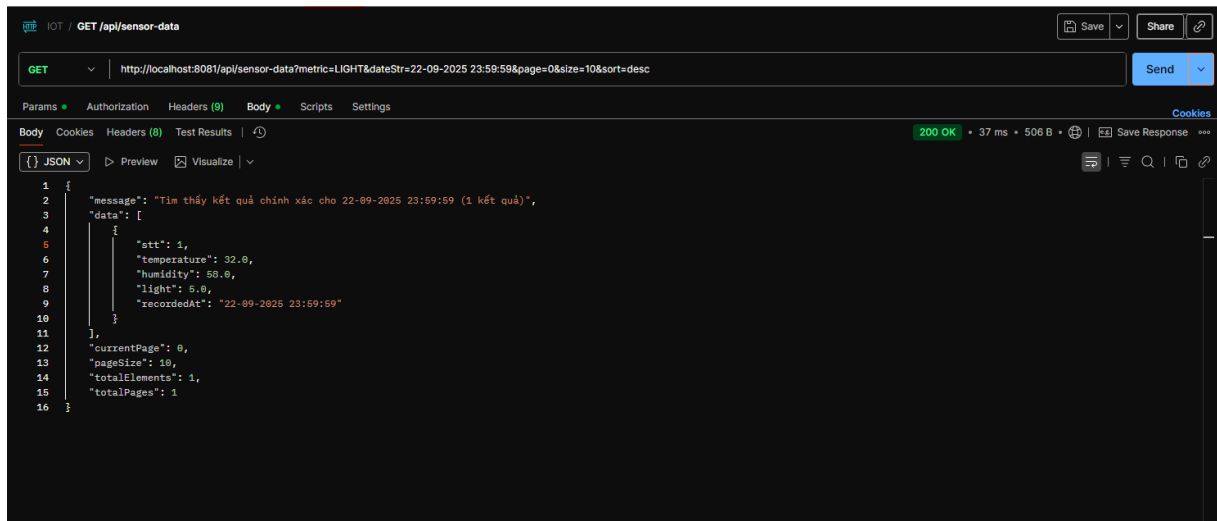
GET http://localhost:8081/api/sensor-data?metric=LIGHT&dateStr=22-09-2025&page=0&size=10&sort=desc

Params Authorization Headers (9) Body Scripts Settings

Body Cookies Headers (8) Test Results 200 OK • 71 ms • 1.3 KB Save Response

{ } JSON Preview Visualize

```
1 {
2   "message": "Tìm thấy kết quả trong phút 22-09-2025 (10 kết quả)",
3   "data": [
4     {
5       "stt": 1,
6       "temperature": 32.0,
7       "humidity": 58.0,
8       "light": 5.0,
9       "recordedAt": "22-09-2025 23:59:59"
10    },
11    {
12      "stt": 2,
13      "temperature": 32.1,
14      "humidity": 58.0,
15      "light": 5.0,
16      "recordedAt": "22-09-2025 23:59:56"
17    },
18    {
19      "stt": 3,
20      "temperature": 32.6,
21      "humidity": 58.0,
22      "light": 5.0,
23      "recordedAt": "22-09-2025 23:59:53"
24    },
25    {
26      "stt": 4,
27      "temperature": 32.3,
28      "humidity": 58.0,
29      "light": 5.0,
30      "recordedAt": "22-09-2025 23:59:58"
31    },
32    {
33      "stt": 5,
34      "temperature": 32.4,
35      "humidity": 58.0,
36      "light": 5.0,
37      "recordedAt": "22-09-2025 23:59:47"
38    },
39    {
40      "stt": 6,
41      "temperature": 32.4,
42      "humidity": 58.0,
43      "light": 5.0,
44      "recordedAt": "22-09-2025 23:59:44"
45    }
46  ]
47 }
```



POSTMAN IOT / GET /api/sensor-data

GET http://localhost:8081/api/sensor-data?metric=LIGHT&dateStr=22-09-2025 23:59:58&page=0&size=10&sort=desc

Params Authorization Headers (9) Body Scripts Settings

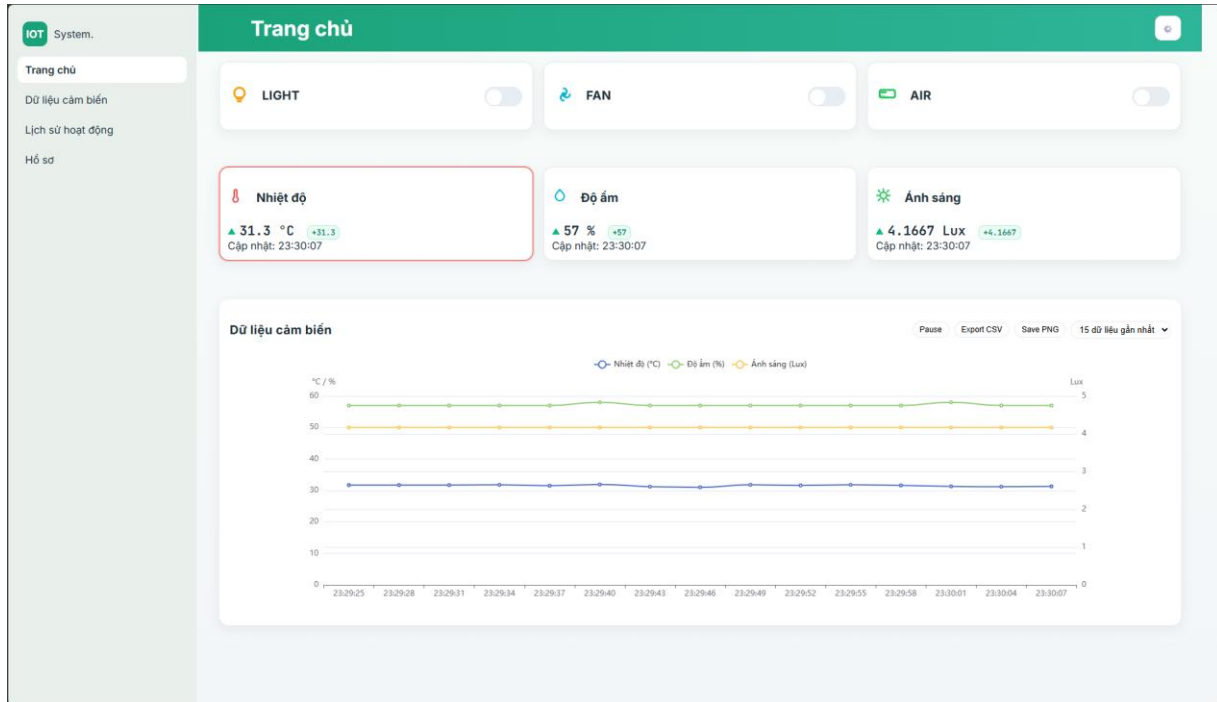
Body Cookies Headers (8) Test Results 200 OK • 37 ms • 506 B Save Response

{ } JSON Preview Visualize

```
1 {
2   "message": "Tìm thấy kết quả chính xác cho 22-09-2025 23:59:59 (1 kết quả)",
3   "data": [
4     {
5       "stt": 1,
6       "temperature": 32.0,
7       "humidity": 58.0,
8       "light": 5.0,
9       "recordedAt": "22-09-2025 23:59:59"
10    }
11  ],
12  "currentPage": 0,
13  "pageSize": 10,
14  "totalElements": 1,
15  "totalPages": 1
16 }
```

4. Giao diện

4.1. Giao diện trang chủ



4.2. Giao diện dữ liệu cảm biến

The 'Dữ liệu cảm biến' (Sensor Data) section of the IOT System. It features a search bar, sorting options (Sắp xếp: Mới nhất, Tắt cả), and a 'Tìm kiếm' button. Below the search bar are links for 'CSV' and 'Hiện thị 15 bản ghi'. The data is presented in a table with 5 columns: STT, Nhiệt độ (°C), Độ ẩm (%), Ánh sáng (Lux), and Thời gian. The table shows 15 rows of data, with the first 14 rows having a 'Thời gian' value and the 15th row having a 'Thời gian' value of '25-09-2025 23:29:25'. The table also includes a 'Hiện thị 15 bản ghi (trang 1/339)' and a pagination control showing '1' of 6 pages.

STT	Nhiệt độ (°C)	Độ ẩm (%)	Ánh sáng (Lux)	Thời gian
1	31.3	57.8	4.2	25-09-2025 23:30:07
2	31.2	57.8	4.2	25-09-2025 23:30:04
3	31.3	58.8	4.2	25-09-2025 23:30:01
4	31.6	57.8	4.2	25-09-2025 23:29:58
5	31.8	57.8	4.2	25-09-2025 23:29:55
6	31.6	57.8	4.2	25-09-2025 23:29:52
7	31.8	57.8	4.2	25-09-2025 23:29:49
8	31.8	57.8	4.2	25-09-2025 23:29:46
9	31.2	57.8	4.2	25-09-2025 23:29:43
10	31.9	58.8	4.2	25-09-2025 23:29:40
11	31.5	57.8	4.2	25-09-2025 23:29:37
12	31.8	57.8	4.2	25-09-2025 23:29:34
13	31.7	57.8	4.2	25-09-2025 23:29:31
14	31.7	57.8	4.2	25-09-2025 23:29:28
15	31.7	57.8	4.2	25-09-2025 23:29:25

